VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DETEKCE SÍŤOVÝCH ÚTOKŮ POMOCÍ NÁSTROJE TSHARK

DETECTION OF NETWORK ATTACKS USING TSHARK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JINDŘICH DUDEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN HOLKOVIČ**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/20984/2017/xdudek04

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Dudek Jindřich, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Detekce síťových útoků pomocí nástroje Tshark**
Detection of Network Attacks Using Tshark
Kategorie: Počítačové sítě

Pokyny:

1. Pro vybrané síťové útoky a nastudujte možnosti jejich detekce.
2. Navrhněte detektor útoků využívající nástroj Tshark se zaměřením na snadné přidávaní nových útoků.
3. Návrh implementujte tak, aby bylo možný jednotlivé útoky zadávat deklarativně. Zároveň musí systém umožňovat jednoduché přidávaní nových útoků.
4. Implementaci otestujte na vhodných testovacích souborech, zaměřte se i na falešnou detekci (false positivies).
5. Zhodnoďte výsledky.

Literatura:

- DeFino, Steven, and Larry Greenblatt. Official Certified Ethical Hacker Review Guide: For Version 7.1. Cengage Learning, 2012.
- Canavan, John E. Fundamentals of network security. Artech House, 2001.
- Tulloch, Mitch. Microsoft encyclopedia of security. Microsoft Press,, 2003.
- Sanders, Chris. Practical packet analysis: Using Wireshark to solve real-world network problems. No Starch Press, 2017.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2 ze zadaní.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Holkovič Martin, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Diplomová práce se zabývá návrhem a implementací nástroje pro detekci síťových útoků ze zachycené síťové komunikace. K tomu využívá paketový analyzátor tshark, jehož význam spočívá v převedení vstupního souboru se zachycenou komunikací do textového formátu PDML, přičemž účelem této konverze je flexibilnější zpracování vstupních dat. Při návrhu nástroje je kladen důraz na jeho rozšiřitelnost o detekci nových síťových útoků a jejich snadnou integraci. Z tohoto důvodu je součástí práce také navržení komplexního deklarativního zápisu síťových útoků v serializačním formátu YAML. Ten umožňuje specifikovat klíčové vlastnosti síťových útoků a podmínky pro jejich detekci. Výsledný nástroj tedy funguje jako interpret navržených deklarativních zápisů, čímž je umožněna jeho rozšiřitelnost o nové typy útoků.

Abstract

This diploma thesis deals with the design and implementation of a tool for network attack detection from a captured network communication. It utilises the tshark packet analyser, the meaning of which is to convert the input file with the captured communications to the PDML format. The objective of this conversion being, increasing the flexibility of input data processing. When designing the tool, emphasis has been placed on the ability to expand it to detect new network attacks and on integrating these additions with ease. For this reason, the thesis also includes the design of a complex declarative descriptions for network attacks in the YAML serialization format. This allows us to specify the key properties of the network attacks and the conditions for their detection. The resulting tool acts as an interpreter of proposed declarative descriptions allowing it to be expanded with new types of attacks.

Klíčová slova

tshark, bezpečnost sítí, síťové útoky, Ping of death, SYN flood, Teardrop, Land, DHCP Spoofing, ARP Spoofing, MAC flooding, ICMP Redirect, útok DAD, RA flood, VLAN hopping, skenování portů, skenování sítě, detektor sítových útoků, deklarativní zápis útoků

Keywords

tshark, network security, network attacks, Ping of death, SYN flood, Teardrop, Land, DHCP Spoofing, ARP Spoofing, MAC flooding, ICMP Redirect, DAD attack, RA flood, VLAN hopping, port scanning, network scanning, network attacks detector, declarative attack description

Citace

DUDEK, Jindřich. *Detekce síťových útoků pomocí nástroje Tshark*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Holkovič

Detekce síťových útoků pomocí nástroje Tshark

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Holkoviče. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jindřich Dudek
20. května 2018

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Ing. Martinu Holkovičovi za jeho ochotu, profesionální přístup, časovou flexibilitu a veškerou poskytnutou pomoc při vedení této diplomové práce.

Obsah

Kapitola 1

Úvod

Internet a počítačové sítě obecně, tak jak je známe dnes, se staly nedílnou a nepostradatelnou součástí lidské populace v civilizovaném světě. Dennodenně nám usnadňují mezilidskou komunikaci, hledání informací či jen představují prostředek pro trávení volného času. Kromě všech výše zmíněných a bezpochyby dalších benefitů, které nám poskytují, je ale nezbytné řešit i problémy v podobě útoků zneužívajících implementačních nedokonalostí síťových protokolů, které zabezpečují běžný chod počítačových sítí. Mnoho z nich již bylo objeveno a podrobně popsáno, včetně vytvoření příslušných záplat pro jejich eliminaci, nicméně stále se objevují útoky nové, proti kterým je nutné hledat detekční a obranné mechanismy. Stále také vznikají nové platformy, technologie a zařízení[1] se síťovou konektivitou, u kterých mohou být opomenuty bezpečnostní aspekty, díky čemuž mohou být zneužity již známé zranitelnosti, případně jejich obdoby.

Základním zabezpečovacím prvkem zajišťujícím ochranu počítačových sítí jsou systémy firewall, jejichž hlavní úlohou je filtrování síťového provozu na základě přesně definovaných pravidel. Některé útoky jsou nicméně natolik sofistikované, že dokáží firewallovou ochranu obejít. Pro komplexnější zabezpečení se tedy využívají tzv. *Intrusion Detection Systémy* (IDS), či *Intrusion Prevention Systémy* (IPS), které aktivně monitorují síťový provoz a snaží se v něm vyhledat, případně blokovat pokusy o síťové útoky.

Cílem této diplomové práce je identifikovat a nastudovat známé síťové útoky, popsat jejich principy a zranitelnosti systémů, kterých je při útocích využíváno, včetně způsobů detekce vybraných útoků. Nejpodstatnější částí je pak na základě těchto informací navrhnout a implementovat komplexní konzolovou aplikaci pro detekci síťových útoků ze zachycené síťové komunikace s využitím nástroje *tshark*. Tento nástroj umožňuje konvertovat zachycenou síťovou komunikaci do textového formátu zvaného *Packet Description Markup Language* (zkráceně *PDML*), což výrazně usnadňuje jeho strojové zpracování a analýzu paketů uvnitř zachycené síťové komunikace.

Hlavní požadavek, na který je při návrhu aplikace kladen důraz, spočívá v její snadné rozšiřitelnosti o detekci nových síťových útoků. Je tedy nezbytné navrhnout takové rozhraní aplikace, které toto umožňuje bez nutnosti výrazné modifikace, či rozšiřování již existujícího kódu.

Diplomová práce je rozdělena do několika kapitol. V kapitole č. 2 je uveden výčet jednotlivých síťových útoků, které je nástroj schopen detekovat, se zaměřením na popis jejich principů a způsobů detekce ze zachycené síťové komunikace.

[1] Např. chytré lednice, televizory, pračky atp., viz http://www.telegraph.co.uk/news/2017/07/24/internet-things-will-leave-home-gadgets-vulnerable-hacks-senior/

Kapitola č. 3 se zabývá návrhem aplikace. Zaměřena je především na popis formátu, který je využit pro deklarativní zápis síťových útoků v jazyce YAML. Dále jsou v ní rozebrány požadavky na aplikaci samotnou a uvedena jejich možná řešení. V kapitole jsou rovněž uvedeny i požadavky na vstupní data.

Další kapitola č. 4 se zabývá samotnou implementací, ve které jsou uvedeny nástroje a knihovny potřebné k běhu nástroje. Zároveň jsou zde uvedeny jednotlivé třídy, jejich vzájemné vztahy a implementační detaily při procesu zpracování jednotlivých typů útoků. V kapitole je dále rozebrán formát výstupu programu a optimalizace, které byly implementovány za účelem snížení paměťové náročnosti nástroje.

Kapitola č. 5 se zabývá testováním funkčnosti a výkonnosti implementovaného nástroje. Popsány jsou jednotlivé metody, které byly použity pro otestování jeho funkčnosti. V závěru kapitoly jsou pak uvedeny výkonnostní charakteristiky implementovaného nástroje a jeho zhodnocení.

Kapitola 2

Principy vybraných síťových útoků

V této kapitole budou popsány vybrané síťové útoky, které je schopen výsledný nástroj detekovat ze zachycené síťové komunikace. Zaměřena bude především na jejich principy, způsoby detekce a případně zranitelnosti, kterých útoky využívají. Informace uvedené v této kapitole jsou podstatné pro návrh deklarativních zápisů jednotlivých síťových útoků, které jsou nezbytné pro zajištění snadné rozšířitelnosti výsledné aplikace.

2.1 Ping of Death

Packet InterNet Groper (zkráceně *ping*) je síťový nástroj umožňující ověřit funkčnost propojení mezi dvěma entitami v počítačové síti, jenž komunikují prostřednictvím rodiny protokolů *TCP/IP*. K tomu využívá zpráv protokolu *ICMP*, který slouží především pro diagnostiku sítě a hlášení chyb při přenosu paketů. Při testování konektivity mezi dvěma entitami je pak ze strany testující stanice odeslána zpráva *Echo request* (typ č. 8) a v případě jejího obdržení zašle testovaná stanice odpověď *Echo reply* (typ č. 0). Pokud testovaná stanice zprávu neobdrží, dojde k zaslání příslušné chybové[1] *ICMP* zprávy [22, 27].

Útok *Ping of Death*, objevený v roce 1996, je variantou útoku typu *Denial of Service* (DoS), který využívá zranitelnosti v návrhu protokolů rodiny *TCP/IP*. Dle standardu *RFC 791* [23] je maximální velikost IP paketu omezena horní hranicí 65 535[2] bytů. Útočník na stanici oběti zašle *ICMP* zprávu typu *Echo Request*, přičemž velikost paketu přesahuje tuto hodnotu. Vzhledem k tomu, že hodnota *MTU*[3] ethernetové technologie je typicky 1500 bytů, musí dojít k fragmentaci tohoto paketu, aby jej bylo možné po médiu přenést. Při obdržení všech těchto fragmentů na cílové stanici se oběť pokusí fragmenty sestavit do původního paketu, přičemž dojde k přetečení paměťového prostoru, který byl pro příchozí paket vyhrazen. Velikost tohoto prostoru odpovídá výše zmíněné hodnotě 65 535 bytů. Podle typu operačního systému či zařízení se tato chyba projeví restartováním nebo pádem cílového systému [4, 27].

Mezi zranitelné systémy patřila většina operačních systémů[4] včetně *Microsoft Windows*, *Mac OS*, či Linuxových distribucí. V dnešní době je proti tomuto útoku již většina užívaných

[1] Typy chybových zpráv viz https://www.onlinecomputertips.com/support-categories/networking/277-ping-error-messages

[2] Tato hodnota vyplývá z faktu, že pole pro určení velikosti paketu zaujímá v IP hlavičce 16 bitů, tzn. maximální velikost paketu je $2^{16} - 1$ bytů.

[3] *Maximum transmission unit* – maximální velikost rámce, kterou je možné v dané technologii na médiu přenést.

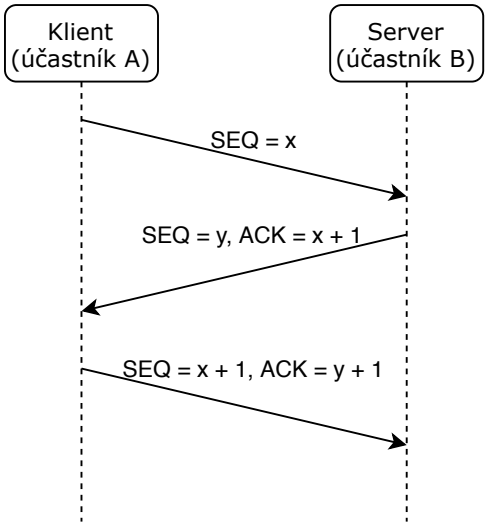
[4] Konkrétní zranitelné verze viz http://insecure.org/sploits/ping-o-death.html

systémů a zařízení odolná. Z výše uvedeného popisu ovšem vyplývá, že zranitelnost se netýkala protokolu *ICMP*, ale způsobu zpracování fragmentovaných *IP* paketů. K realizaci útoku bylo tedy možné využít i jiné protokoly přenášené v rámci protokolu *IP* [27].

Na paketové úrovni je možné tento útok detekovat prostřednictvím kontroly jednotlivých IP fragmentů, pro které musí platit, že součet polí *Fragment Offset* a *Total Length* v jejich hlavičce je větší než výše zmíněná hodnota 65 535. V případě nalezení takového paketu je možné konstatovat, že v průběhu zachytávání síťové komunikace došlo k útoku Ping of Death.

2.2 SYN flood

SYN flood je další z útoků typu *Denial of Service*, který využívá zranitelnosti spojově orientovaného protokolu *TCP*. Na počátku každé *TCP* komunikace dochází k procesu navázání spojení, tzv. *three-way handshaku*, jehož účelem je zajištění spolehlivého doručování paketů. Průběh navázání spojení je znázorněn na obrázku č. 2.1. V první části účastník *A* pošle zprávu *SYN* protistraně *B* s náhodně zvoleným sekvenčním číslem *x*. V reakci na přijetí této zprávy účastník *B* odpoví zprávou *SYN+ACK*, ve které zasílá své náhodně vygenerované číslo *y* a zároveň potvrdí protistraně, že očekává následující zprávu se sekvenčním číslem *x+1*. V poslední fázi pak účastník *A* reaguje potvrzující zprávou *ACK* s číslem *y+1*, po jejímž přijetí protistranou je spojení navázáno [4, 13].



Obrázek 2.1: Sekvenční diagram znázorňující navázání spojení u *TCP* protokolu.

Princip útoku pak spočívá v periodickém zasílání *SYN* paketů směrem k oběti, ve kterých je uvedena nevalidní zdrojová *IP* adresa tak, aby s touto adresou nebylo možné ustavit spojení. Oběť pak reaguje zasláním zprávy *SYN+ACK* zpět na podvrženou adresu, ze které už ale není zaslána poslední zpráva *ACK*, která by završila proces navazování komunikace. Tím vznikne napůl otevřené *TCP* spojení, u něhož nikdy nedojde k jeho kompletnímu ustavení, nicméně na stanici oběti jsou pro něj rezervovány zdroje. Rychlé periodické zasílání dalších *SYN* zpráv s různými podvrženými *IP* adresami pak vede ke kompletnímu vyčerpání těchto zdrojů, díky čemuž oběť není schopná přijímat spojení od legitimních uživatelů, kterým je tak znemožněna komunikace s obětí. U některých typů systémů navíc může dojít

ke kompletnímu vyčerpání systémové paměti, což vede k následnému pádu celého systému [4, 10].

Obranou proti tomuto útoku může být například snížení hodnot časovačů určujících po jakou dobu mohou existovat napůl otevřená spojení a po vypršení tohoto časového limitu je uzavřít. Dalším řešením je použití *IPS*, či *IDS* systémů, které útok detekují a případně zablokují [4, 27].

Detekovat útok ze zachycené síťové komunikace je možné pomocí kontroly zasílaných paketů mezi dvěma síťovými entitami (identifikovanými IP adresou) na konkrétních portech prostřednictvím transportního protokolu TCP. Pokud je nalezeno větší množství síťových toků, které sestávají pouze z paketů *SYN* a *SYN+ACK*, je možné počet těchto toků porovnat s hodnotou stanoveného prahu[5] a při jeho překročení je možné prohlásit útok za detekovaný.

2.3 Teardrop

Dalším útokem z rodiny *DoS* útoků je tzv. *Teardrop*, objevený v roce 1997. Podobně jako *Ping of Death* (popsaný v kapitole 2.1) využívá zranitelnosti při sestavování fragmentovaných dat do původního datagramu na cílové stanici oběti.

Každý fragment má uvnitř sebe obsaženou informaci o jeho pozici v původním nefragmentovaném datagramu. Jedná se o pole s názvem *Fragment offset* o celkové délce 13 bitů, které je definované protokolem *IP* [23]. Útočník zašle na stanici obětí sérii fragmentovaných datagramů, ve které první z nich má offset nastavený na hodnotu 0. U dalších fragmentů tuto hodnotu nastaví tak, aby bylo jisté, že se jednotlivé datagramy budou překrývat[6] s těmi předcházejícími [16, 34]. Při pokusu o sestavení překrývajících se datagramů pak dojde, kvůli implementační chybě v sestavovacím algoritmu, k neoprávněnému přístupu do paměti, což vyústí v pád, či zamrznutí cílového systému.

Tato zranitelnost se týkala především systému *MS Windows 95*, či Linuxových distribucí s verzí jádra nižší než *2.1.63*. Po opravě této zranitelnosti se objevily i další útoky využívající podobný princip, jako je například *Teardrop2*, *Newtear* a další [27].

Pro detekci tohoto útoku poskytuje prostředky samotný nástroj *tshark*, který v případě nalezení překrývajících se datagramů rozšíří jejich hlavičky o pole s názvem *ip.fragment.overlap* a nastaví indikační bit na hodnotu logické jedničky. V datagramech tedy stačí kontrolovat přítomnost tohoto pole a v případě jeho nalezení je možné konstatovat detekci útoku.

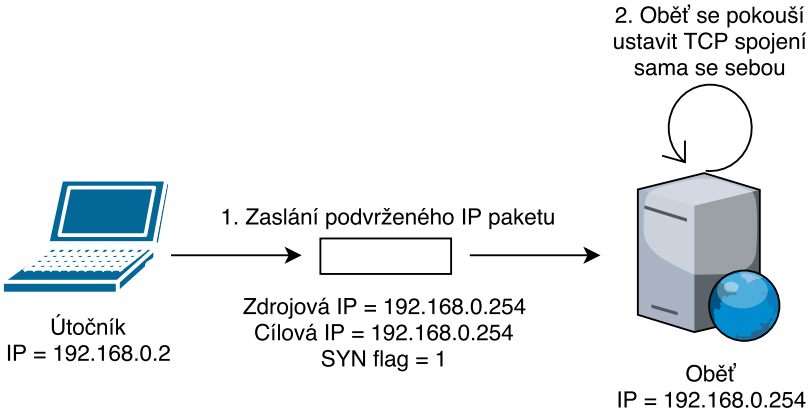
2.4 Land

Local Area Network Denial (zkráceně *LAND*) je dalším ze zástupců skupiny útoků *DoS*. Byl objevený v roce 1997 a jeho princip spočívá ve vytvoření podvržených *IP* paketů tak, aby se cílová stanice oběti pokoušela ustavit *TCP* spojení sama se sebou. Toho je docíleno zasláním paketu, v němž je uvedena identická zdrojová i cílová *IP* adresa, jež odpovídá *IP* adrese oběti. Zároveň je v paketu nastaven i *SYN* flag, který indikuje požadavek na ustavení nového spojení. Přijetí tohoto paketu obětí pak podle typu zařízení vyústí ve zpomalení

[5] Tento práh může být zvolen dle charakteristiky dané sítě, ve které k zachytávání síťové komunikace dochází.

[6] K překryvu dochází, pokud se součet offsetu a délky datagramu odlišuje od hodnoty offsetu následujícího datagramu, viz. https://security.radware.com/ddos-knowledge-center/ddospedia/teardrop-attack/.

chodu systému, jeho zamrznutí, či pádu [14, 27]. Výše popsaný princip je ilustrován na obrázku č. 2.2.



Obrázek 2.2: Ilustrace principu útoku Land.

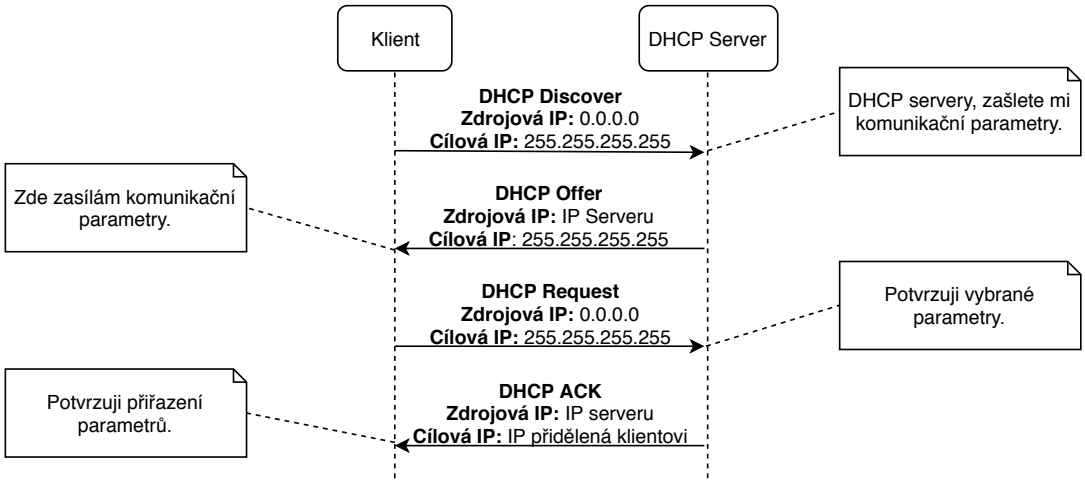
Jev, že pakety mají stejnou zdrojovou a cílovou *IP* adresu, není nic neobvyklého u uživatelských stanic, na kterých mohou běžet různé služby[7], k nimž se uživatel z dané stanice lokálně připojuje. Pokud se ale podobný paket objeví na jiném místě v síti, je možné konstatovat, že se jedná o útok *LAND*. Tato zranitelnost byla původně objevena u operačního systému *MS Windows 95*, nicméně záhy byla zjištěna i u Unixových systémů, *Cisco* routerů, či různých druhů síťových tiskáren [27]. Detekce tohoto útoku je možná porovnáváním zdrojové a cílové *IP* adresy u paketů, které mají nastavený příznak *SYN*. V případě shody je pak za předpokladu, že síťová komunikace nebyla zachycena na uživatelské stanici, možné prohlásit útok za detekovaný.

2.5 DHCP spoofing

Dynamic host configuration protocol (zkráceně *DHCP*) [7] je protokol, který umožňuje dynamické poskytování *IP* adresy a dalších informací[8], nezbytných pro komunikaci v síti, nově připojeným zařízením. Proces iniciuje klient, který do sítě zašle broadcastovou zprávu *DHCP Discover*. Tím vyzve *DHCP* server, aby reagoval zprávou *DHCP Offer*, ve které mu zašle komunikační informace zmíněné výše. Klient může obdržet více těchto odpovědí od různých serverů, vybere si z nich tedy tu první přijatou. Reakcí je pak zaslání broadcastové zprávy *DHCP Request*, kterou potvrdí výběr *DHCP* serveru a přiřazených informací. Poslední zprávou *DHCP ACK* pak server potvrdí klientovi jeho výběr, čímž je konfigurace ukončena [29]. Výše popsaný proces, včetně využívaných *IP* adres u konkrétních zpráv, je ilustrován na obrázku č. 2.3.

Útok *DHCP Spoofing* pak zneužívá této konfigurace k provedení *Man in the middle* (*MiTM*) útoku tak, že útočník podvrhne zprávu *DHCP Offer*, ve které zašle oběti vlastní komunikační parametry, včetně svojí *IP* adresy jako adresu výchozí brány, díky čemuž je schopen přesměrovat síťovou komunikaci od oběti směrem k sobě. Aby proces podvrhnutí proběhl úspěšně, je nezbytné, aby oběť přijala útočníkovu zprávu jako odpověď legitimního

[7] Např. HTTP server, databázový server, apod.
[8] Mezi tyto informace patří např. maska sítě, adresa výchozí brány, *DNS* server atp.



Obrázek 2.3: Sekvenční diagram znázorňující dynamickou konfiguraci komunikačních parametrů pomocí protokolu *DHCP*.

serveru. Jednou z možností, jak toho docílit, je zaslat zprávu dříve než skutečný *DHCP* server. Tento způsob je ale poměrně nespolehlivý obzvlášť v případě, pokud stanice oběti byla již dříve v síti připojena. Další možností je záměrně vyčerpat všechny *IP* adresy poskytované skutečným *DHCP* serverem, který následně přestane na požadavky klientů reagovat. Třetí možností spočívá ve vyřazení *DHCP* serveru z provozu např. pomocí *DoS* útoku [31].

Nevýhoda tohoto útoku spočívá v tom, že útočník může sledovat pouze pakety směřující od oběti na výchozí bránu, nikoliv pakety směřující z výchozí brány na stanici oběti. Útočník tedy může vidět pouze jednosměrnou síťovou komunikaci, protože oklamal jen oběť a nikoliv výchozí bránu. Odposlouchávání obousměrné komunikace je možné docílit např. tak, že útočník bude vykonávat překlad adres NAT[9], či podvrhne oběti falešný *DNS* server.

Pro korektní detekci útoku *DHCP Spoofing* je zapotřebí informace o celkovém počtu *DHCP* serverů nacházejících se v síťovém segmentu, ve kterém byla síťová komunikace zachycena. Aby bylo možné mezi sebou asociovat jednotlivé zprávy *DHCP* protokolu, je v hlavičce paketů uvedena položka *Transaction ID*, jejíž hodnota je stejná pro všechny zprávy v rámci celého procesu dynamické konfigurace IP adresy. Detekce útoku pak spočívá v nalezení celkového počtu zpráv *DHCP Offer* s identickou hodnotou pole *Transaction ID*. Tento počet je porovnán se známým počtem *DHCP* serverů v síťovém segmentu a pokud je zpráv *DHCP Offer* nalezeno více, musela být alespoň jedna z nich zaslána nelegitimním *DHCP* serverem.

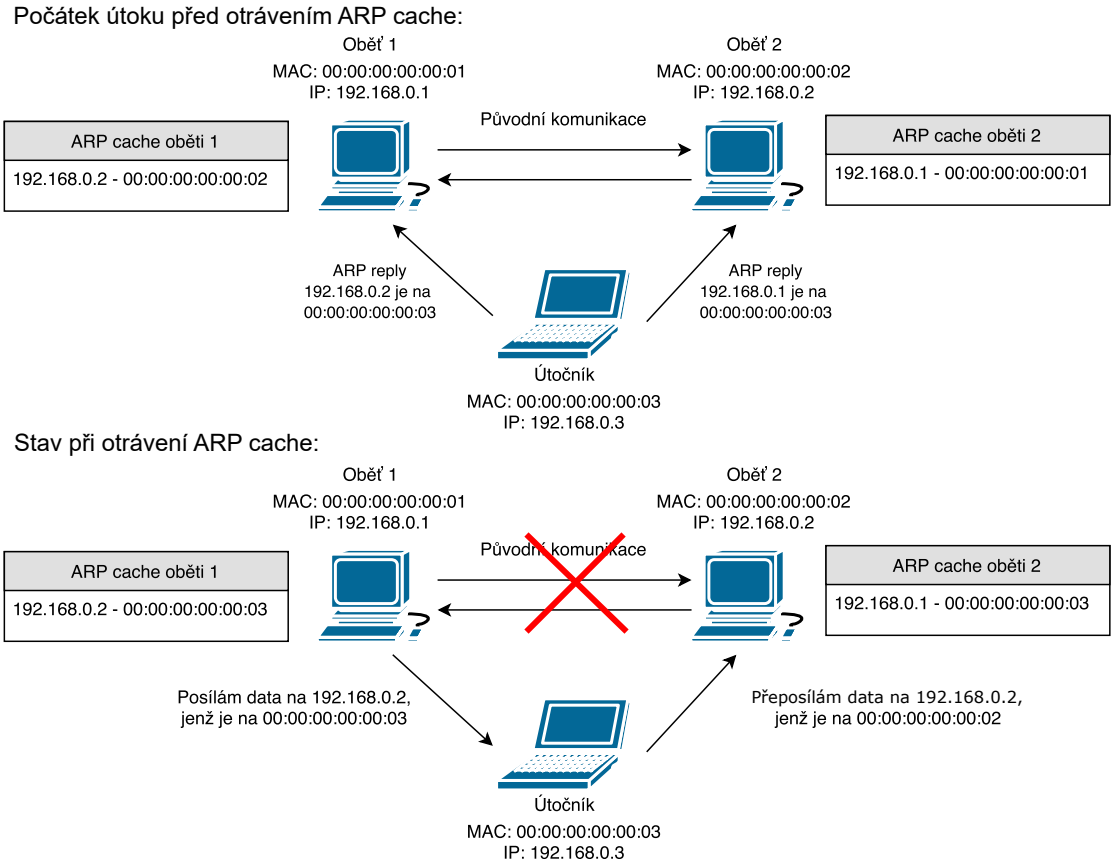
2.6 ARP Spoofing

Address Resolution Protocol (*ARP*) [21] je protokol tvořící propojení mezi linkovou a síťovou vrstvou modelu *TCP/IP*. Slouží k získání *MAC* adresy síťového rozhraní v lokální síti pomocí známé *IP* adresy. Kdykoliv spolu chtějí dvě entity v rámci lokální sítě komunikovat, musí odesilatel nejprve zjistit *MAC* adresu cílového rozhraní. Dojde tedy k zaslání zprávy *ARP request*, jejímiž příjemci jsou všechna zařízení v rámci daného síťového segmentu. Zařízení, jemuž patří dotazovaná *IP* adresa, pak odpovídá odesilateli prostřednictvím uni-

[9] Network address translation.

castové zprávy *ARP reply*, v níž je obsažena požadovaná *MAC* adresa cílového rozhraní. Zjištěné mapování mezi *IP* a *MAC* adresou se poté uchovává v tzv. *ARP* cache, aby se redukoval počet zasílaných *ARP requests* [20].

Při útoku *ARP Spoofing* (taktéž *ARP cache poisoning*) se využívá faktu, že zdrojová *IP* adresa, uvedena v hlavičce paketu, nemusí být skutečná adresa zařízení, které paket zaslalo. Protokol *ARP* je bezstavový, tudíž lze zaslat odpověď bez předcházejícího dotazu. V podvržené zprávě tedy útočník asociuje svojí *MAC* adresu s *IP* adresou jedné z obětí a tuto zprávu zašle druhé oběti, čímž do její cache vloží podvržený záznam a veškerá komunikace od oběti na danou *IP* adresu je tím přesměrována k útočníkovi. Aby odposlouchávání komunikace mezi dvěma entitami probíhalo obousměrně, musí útočník tento proces zopakovat i pro druhou z obětí. K zajištění úspěšnosti útoku je zapotřebí, aby útočník veškerou přijatou komunikaci přeposílal směrem k původnímu příjemci, jinak by mohlo dojít k jeho odhalení. Vzhledem k tomu, že záznamy v *ARP* cache po určité době expirují, musí útočník zajistit opakované zasílání *ARP* odpovědí tak, aby udržel falešnou informaci o *MAC* adresách v cache pamětech komunikujících stran [8]. Na obrázku č. 2.4 je demonstrován princip útoku. V první polovině obrázku je uveden stav *ARP* cache ještě před jejich otrávením, v druhé pak přeposílání komunikace po provedení útoku.



Obrázek 2.4: Princip útoku *ARP* Spoofing.

Pro detekci útoku je možné využít prostředků nástroje *tshark*, který je schopen detekovat prostřednictvím analýzy zpráv ARP reply situaci, kdy s jednou MAC adresou je asociováno více IP adres. Tato skutečnost je signalizována přítomností pole s názvem *arp.duplicate-*

address-frame s hodnotou nastavenou na logickou jedničku v hlavičce rámce. Pokud žádný takový rámec není nalezen, je navíc zapotřebí ověřit poměr počtu zpráv *ARP reply* a *ARP request*, protože každé odpovědi by mělo předcházet zaslání žádosti. Pokud je počet odpovědí výrazně vyšší, než počet žádostí, je možné taktéž konstatovat, že se s největší pravděpodobností jedná o útok. Vypočítaný poměr se tedy porovná s hodnotou určeného prahu a na základě výsledku je učiněn závěr.

2.7 MAC flooding

MAC flooding je v síťovém prostředí typ útoku, který využívá zranitelnosti ve fungování přepínačů (tzv. switchů). Switch je síťové zařízení, které rozesílá příchozí rámce pouze na konkrétní port podle cílové *MAC* adresy[10]. Každý přepínač si ve své paměti udržuje tzv. *CAM* tabulku a na základě jejího obsahu rozhoduje, na jaký cílový port rámec zaslat. V tabulce je tedy uvedeno mapování mezi *MAC* adresou cílového rozhraní a portem switche. Pokud switch příjme rámec, dojde k porovnání jeho cílové *MAC* adresy se záznamy *MAC* adres v tabulce. Jestliže je nalezena shoda, dojde k zaslání rámce na příslušný korespondující port. V opačném případě je rámec rozeslán na všechny dostupné porty s výjimkou toho, ze kterého byl přijat [32].

Proces vytváření záznamů v *CAM* tabulce probíhá během reálného síťového provozu, kdy v případě, že pro zdrojovou *MAC* adresu rámce neexistuje v tabulce žádný záznam, dojde k jejímu přidání do tabulky včetně portu, ze kterého byl rámec přijat. Tyto záznamy mají v *CAM* tabulce omezenou dobu platnosti, po jejímž vypršení je záznam odstraněn, pokud nedojde k jeho obnovení [32].

Při útoku je pak využíván fakt, že *CAM* tabulky mají omezenou kapacitu. Útočník tedy začne periodicky zasílat rámce s náhodně vygenerovanými *MAC* adresami a přepínač pro ně začne vytvářet příslušné záznamy v tabulce. Po určité době dojde k jejímu kompletnímu naplnění, což zapříčiní přepnutí switche do tzv. *fail open* režimu. V tomto stavu není přepínač schopen vkládat do *CAM* tabulky další záznamy, začne se tedy chovat jako hub a přeposílat příchozí rámce na všechny své porty, kromě těch příchozích. To umožní útočníkovi, jehož zařízení má síťovou kartu nastavenou v promiskuitním režimu, odposlouchávat komunikaci, která mu není určena. K zajištění úspěšnosti útoku je nezbytné generovat dostatečně velké množství rámců v dostatečně krátkém časovém intervalu, aby nedošlo k expiraci falešných záznamů v tabulce a jejímu opětovnému uvolnění [26].

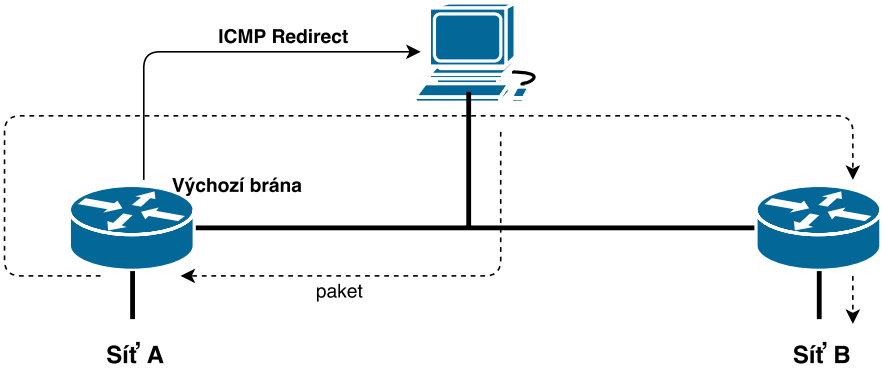
Obrana proti tomuto útoku je možná např. pomocí mechanismu *Port security*, jehož podstata spočívá v omezení počtu unikátních *MAC* adres, které se může switch z konkrétního portu naučit. Při jeho překročení pak nedochází k přidávání dalších záznamů z daného portu do *CAM* tabulky, čímž nedojde k jejímu přeplnění [32].

Detekce útoku *MAC flooding* je poněkud obtížnější a záleží při ní na charakteristikách nástroje, který je pro generování náhodných rámců využit. Některé nástroje totiž při vytváření náhodných *MAC* adres nerespektují pravidlo, že zdrojová *MAC* adresa rámce nesmí být skupinovou adresou. Tuto skutečnost je schopen nástroj *tshark* detekovat a indikuje ji přidáním pole *eth.src_not_group* do hlavičky rámce. Při výskytu většího množství takových rámců je pak pravděpodobné, že se jedná o útok typu *MAC flooding*. Další metodou pro detekci útoku pak může být zjištění celkového počtu unikátních zdrojových *MAC* adres v zachycené komunikaci a tento počet pak porovnat s celkovým počtem zaslaných rámců. Pokud se tento počet příliš neliší, je opět pravděpodobné, že se jedná o útok.

[10] Tím se liší od hubu, který rozesílá příchozí rámce na všechny své porty.

2.8 ICMP Redirect

ICMP Redirect je další zprávou protokolu *ICMP* s kódem č. 5. Tato zpráva byla původně navržena tak, aby ji mohly využívat směrovače k indikaci, že existuje lepší cesta v síti pro zasílání paketů, než je ta, po které jsou pakety aktuálně zasílány. Na obrázku č. 2.5 je možné vidět příklad jejího užití. Ve chvíli, kdy chce koncová stanice komunikovat se zařízením v síti B, zašle paket na výchozí bránu. Výchozí brána ovšem zná lepší cestu do sítě B, zašle tedy zprávu *ICMP Redirect* koncové stanici a přepošle paket na druhý směrovač. Koncová stanice po příjetí zprávy *ICMP Redirect* upraví své směrovací tabulky tak, aby následující pakety směřující do sítě B stanice zasílala přímo na druhý směrovač. Takové chování je typické pro síťové segmenty obsahující více než jeden router [11].



Obrázek 2.5: Příklad užití zprávy ICMP Redirect.

Směrovače a koncové stanice mají přesně definované situace, při kterých by měly zasílat, či přijímat zprávy o přesměrování. Podle *RFC 1122* [3] by koncové stanice zprávu *ICMP Redirect* generovat vůbec neměly, nicméně v původním návrhu tato situace není implementačně ošetřena. Tohoto faktu může útočník využít k vytvoření podvrženého paketu, který zašle na stanici oběti pod záminkou, že byla vygenerována některým ze směrovačů. Tím může modifikovat směrovací tabulku oběti a způsobit tím útok *DoS*[11] nebo *MiTM* [11].

Současné operační systémy se snaží proti tomuto typu útoku implementovat různé obranné mechanismy, podobně jako samotné směrovače. Mezi ně patří kupříkladu ignorace tohoto typu zpráv, pokud jsou zaslány z neznámých síťových rozhraní, či důkladná kontrola routerů v daném síťovém segmentu, zda jsou nakonfigurovány pro zasílání tohoto typu zpráv [11].

Detekce útoku spočívá ve vyhledávání zpráv typu *ICMP Redirect* v zachycené síťové komunikaci. Vzhledem k tomu, že se nemusí vždy jednat o útok a zpráva může být legitimně zaslána některým ze směrovačů, nelze útok detekovat s naprostou určitostí. O nalezení zprávy je ovšem zapotřebí podat hlášení a síťový administrátor pak může na základě znalosti o konfiguraci směrovačů učinit závěr.

[11] Například vložením neexistujícího next-hopu do směrovací tabulky.

2.9 Duplicate Address Detection attack

Mechanismus *Duplicate Address Detection* (*DAD*) je využíván při procesu bezstavové autokonfigurace *IPv6* adresy (angl. *SLAAC*) k určení, zda nově vygenerovanou *IPv6* adresu nemá již nakonfigurovanou jiné zařízení v daném síťovém segmentu. Proces bezstavové autokonfigurace *IPv6* adresy je zahájen zasláním zprávy *Router Solicitation* protokolu *ICMPv6* od nově připojeného zařízení v síti. Směrovače při jejím přijetí reagují zasláním zprávy *Router Advertisement*, v níž je obsažen síťový prefix dané sítě. Kombinací prefixu a vytvořeného identifikátoru rozhraní (angl. *interface identifier*) pak zařízení generuje 128 bitů dlouhou *IPv6* adresu [5, 25].

V další fázi je nutné ověřit unikátnost vygenerované *IPv6* adresy pomocí mechanismu *DAD*. Tento proces je nezbytnou součástí každé bezstavové autokonfigurace. Zařízení vygeneruje multicastovou zprávu protokolu *ICMPv6* s názvem *Neighbor Solicitation*, která je zaslána všem zařízením v daném síťovém segmentu. Uvnitř této zprávy je uvedena *IPv6* adresa, jejíž duplicitní užívání je zapotřebí zkontrolovat. Pokud některé zařízení tuto adresu používá, zareaguje zasláním zprávy *Neighbor Advertisement*, čímž dá dotazující stanici najevo, že adresu nelze použít. Proces vytváření nové *IPv6* adresy se tedy celý opakuje, až do chvíle, kdy zařízení neobdrží žádnou odpověď, což indikuje, že vygenerovaná adresa je unikátní a je možné ji nakonfigurovat jako adresu rozhraní [25].

Podstata útoku pak spočívá v tom, že útočník nedovolí žádné nově připojené stanici v síti nakonfigurovat *IPv6* adresu, díky čemuž ji znemožní *IPv6* síťovou konektivitu. K tomu útočník využívá implementační nedokonalosti výše uvedeného *DAD* mechanismu tak, že na každý pokus o ověření unikátnosti vygenerované *IPv6* adresy pomocí *Neighbor Solicitation* reaguje podvrženou zprávou *Neighbor Advertisement*. Každá nová adresa, kterou tedy zařízení vygeneruje, bude útočníkem prohlášena za duplicitní a stanici se tak nikdy nepodaří nakonfigurovat *IPv6* adresu síťového rozhraní. V podstatě se tedy jedná o útok *Denial of Service* na *IPv6* konektivitu nově přípojených stanic v síti [25, 28].

Obrana proti tomuto typu útoku je poměrně komplikovaná, protože základní protokol *Neighbor Discovery* žádný obranný mechanismus neposkytuje. Z tohoto důvodu bylo vytvořeno bezpečnostní rozšíření s názvem *Secure Neighbor Discovery* (*SEND*), definováno v *RFC 3971* [1]. Obrana je založena na mechanismu kryptograficky generovaných adres[12] (angl. *Cryptographically Generated Addresses*), jež jsou vytvářeny pomocí kryptografické hashovací funkce. Výše uvedený mechanismus je zcela nezávislý na mechanismu *IPSec*, jenž byl původně pro tento účel navržen [30].

Detekce útoku v zachycené síťové komunikaci je založena na vyhledání zpráv *Neighbor Solicitation*, jejichž zdrojová *IPv6* adresa odpovídá ve zkráceném zápisu hodnotě „::“[13], a jim odpovídajících zpráv *Neighbor Advertisement*. Tyto zprávy lze navzájem asociovat pomocí pole s názvem *Target Address*, jejichž hodnota je u obou zpráv identická. Počet těchto nálezů je pak porovnán s hodnotou prahu a v případě jeho překročení je útok prohlášen za detekovaný. Při identifikaci tohoto typu útoku je využíván fakt, že vzhledem k velkému adresovému *IPv6* prostoru je vznik většího množství duplicitních adres velice nepravděpodobný.

[12] Podrobnější popis viz. https://en.wikipedia.org/wiki/Cryptographically_Generated_Address

[13] Tato zdrojová *IPv6* adresa značí, že zařízení nemá doposud nakonfigurovanou adresu.

2.10 Router Advertisement flood

Nedílnou součástí protokolu *ICMPv6* je zpráva *Router Advertisement* (*RA*) s typem č. 134. Směrovače tuto zprávu periodicky zasílají všem uzlům v daném síťovém segmentu[14], čímž signalizují svojí přítomnost a zároveň poskytují informace nezbytné pro komunikaci v síti (např. síťový prefix, metodu vytvoření *IPv6* adresy, atp.). Současně je zasílána i v reakci na zprávu *Router Solicitation*, která se uplatňuje v bezstavové autokonfiguraci *IPv6* adresy (tzv. *SLAAC*). Kdykoliv uzel zprávu *RA* obdrží, aktualizuje si svojí směrovací tabulku a v případě nastavení příznaku autokonfigurace taktéž vytvoří novou *IPv6* adresu s prefixem uvedeným uvnitř zprávy [12, 28].

Při samotném útoku je využíván fakt, že síťové uzly mohou mít nakonfigurováno více *IPv6* adres pro konkrétní rozhraní a nejsou schopny rozlišit legitimní zprávu *RA* od podvržené [18]. Útočník tedy začne do sítě zasílat velké množství podvržených paketů *RA* s náhodně vygenerovanými síťovými prefixy. Jejich přijetí pak na straně ostatních uzlů v síti vyvolá proces vytváření nových *IPv6* adres a aktualizaci směrovacích tabulek. Velké množství těchto operací v krátkém časovém úseku pak vede k maximálnímu vytížení procesoru a operační paměti, díky čemuž přestane stanice reagovat na uživatelovy požadavky a jediným východiskem je její restartování. Cílová adresa podvržených paketů *RA* je nastavena na multicastovou adresu *ff02::1*, jedná se tedy o útok *Denial of Service* postihující všechny zranitelné systémy v celé lokální *IPv6* síti [33].

Na tento typ útoku nejsou náchylné všechny operační systémy. Se záplavou *RA* paketů jsou schopny se vyrovnat operační systémy Linux, zatímco velkým potížím čelí systémy od *Microsoft Windows* [33]. Obrana proti útoku je poměrně komplikovaná. Využít lze filtrování podvržených zpráv *RA* pomocí různých mechanismů, mezi něž patří např. *firewall*, *Router Advertisement Guard*, či *Acess Control Lists*. Dalším řešením je pak využití protokolu *SEND*, zmíněného na konci kapitoly 2.9, či aplikace standardu *IEEE 802.1x*. Detailnější informace o těchto metodách a další způsoby zabezpečení je možné nalézt v diplomové práci od K. Mudaliar [18].

Pro korektní odhalení útoku *RA flood* ze zachycené síťové komunikace je zapotřebí znalost o celkovém počtu síťových *IPv6* prefixů užívaných v dané síti. Při detekci je pak ze všech zpráv *Router Advertisement* získán celkový počet unikátních síťových prefixů, které byly ve zprávách obsaženy. Pokud tento počet nesouhlasí se známým počtem standardně užívaných prefixů v dané síti, je možné konstatovat, že se jedná o síťový útok *RA flood*.

2.11 VLAN hopping

Virtual LAN (*VLAN*) je prostředek využívaný pro rozdělení sítě na logické části nezávisle na jejím fyzickém uspořádání. K jejímu vytvoření obvykle dochází nakonfigurováním tzv. VLAN přepínače, pomocí kterého lze rovněž jednotlivé VLAN sítě propojit. Používány jsou především z důvodu, že umožňují zmenšovat broadcastovou doménu, snadněji lokalizovat problémy v síti a rovněž usnadňují správu síťových zařízení [2].

VLAN taktéž poskytuje prostředky, které mohou sloužit jako bezpečnostní mechanismus pro filtrování síťového provozu mezi jednotlivými VLAN sítěmi. Toto filtrování nicméně nemusí být vždy úplně funkční a za určitých podmínek je možné jej obejít [15]. V následujících podkapitolách budou uvedeny mechanismy pro realizaci útoku *VLAN hopping*, který

[14] Na multicastovou adresu *ff02::1*.

umožňuje navázat neautorizovanou komunikaci se síťovými zařízeními napříč sítěmi VLAN, jenž by za normálních podmínek byla nedostupná.

Pro správné porozumění níže uvedeným principům je nejdříve nezbytné vysvětlit několik pojmů. Prvním z nich je tzv. *trunk port*, který se používá pro označení portu zařazeného do více VLAN sítí. Vzájemné spojení dvou trunk portů se pak nazývá *trunk link*. Dalším pojmem je tzv. *nativní VLAN*[15], což je taková virtuální LAN síť, do které přepínač zařadí všechny neoznačkované rámce, které příjme na svůj trunk port (pojem značkování viz. následující kapitola 2.11.1). Vždy platí, že nativní VLAN musí být nastavena shodně na obou stranách trunk portu. Veškerý provoz, který je zařazen do nativní VLAN, se při přenosu neoznačuje tagem [2].

2.11.1 Double tagging

Metoda nazývaná *Double tagging* využívá zranitelnosti protokolu 802.1Q a způsobu, jakým jsou zpracovávány jednotlivé rámce pomocí přepínačů v sítích VLAN. Vzhledem k tomu, že jednotlivé rámce mohou být prostřednictvím trunk linků přenášeny mezi více VLAN sítěmi, je zapotřebí u nich uvést informaci, z jaké VLAN sítě pochází. Standard 802.1Q definuje mechanismus, který pro označení rámců využívá speciální značku (tzv. *tag*) o velikosti 4 bytů, umístěnou v jejich hlavičce. Pomocí ní je jednoznačně identifikována síť VLAN, ze které rámec pochází. Tento tag je k rámci připojen na odchozím trunk portu. Na přijímací straně trunku je pak tento tag odstraněn a paket je zaslán na odpovídající port podle identifikátoru VLAN sítě [15].

Pro úspěch metody *Double tagging* musí být útočník připojen k rozhraní, které patří do nativní VLAN sítě trunk portu. Při provádění útoku pak útočník změní původní rámec tak, že do jeho hlavičky vloží dva VLAN tagy. První z nich (vnější) identifikuje VLAN síť útočníka a druhý (vnitřní) síť oběti. První přepínač, na který dvojitě označkovaný rámec dorazí, detekuje pouze vnější tag označující VLAN síť, do které rozhraní útočníka skutečně patří a následně tento tag odstraní. Rámec je pak dále zaslán na všechny porty patřící do nativní VLAN, přičemž jedna z jeho kopií je přeposlána po trunk linku, na jehož konci se vyskytuje přepínač, jehož prostřednictvím lze proniknout do VLAN oběti. Tento přepínač následně detekuje druhý (vnitřní) tag rámce a na základě jeho hodnoty ho přepošle do sítě oběti, čímž je útok dokonán [15]. Mezi jeho charakteristiky patří především fakt, že je pouze jednosměrný, tzn. na zaslaný rámec útočníkovi nikdy nedorazí odpověď. Výše popsaný princip je ilustrován na obrázku 2.6.

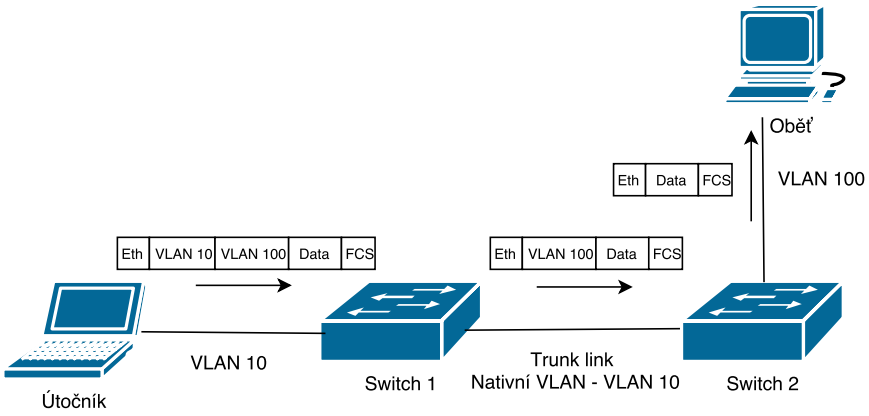
Detekce útoku ze zachycené síťové komunikace je poměrně přímočará. U každého rámce je zapotřebí kontrolovat hlavičku, zda obsahuje více než jeden VLAN tag. V kladném případě je zapotřebí vypsat varovné hlášení o potenciální detekci. Pouze varovné hlášení je vypisováno z toho důvodu, že se vždy nemusí jednat o síťový útok, což je závislé na konfiguraci VLAN sítě. O útok se nejedná například v případě, kdy poskytovatel internetového připojení (tzv. *ISP*) používá metodu tunelování *Q-in-Q*[16].

2.11.2 Switch spoofing

Tento typ útoku zneužívá protokol *Dynamic Trunking Protocol* (*DTP*), který je určen pro dynamické vytváření trunk linků mezi dvěma VLAN sítěmi. Každý port přepínače, který

[15] Viz. http://www.omnisecu.com/cisco-certified-network-associate-ccna/what-is-native-vlan.php

[16] Viz. https://www.juniper.net/documentation/en_US/junos/topics/concept/qinq-tunneling-qfx-series.html



Obrázek 2.6: Ilustrace útoku Double tagging.

podporuje VLAN, se nachází v jednom z módů (angl. tzv. *trunking mode*), jenž určují, zda je v daném módu možné dynamicky ustavit nový trunk. Jejich přehled, včetně popisu a uvedení zranitelnosti vůči útoku *Switch spoofing* je možné nalézt v tabulce 2.1. Z této tabulky vyplývá, že pro realizaci útoku je nezbytné, aby port vedoucí směrem ke stanici útočníka byl v jednom ze stavů *dynamic desireable*, *dynamic auto* nebo *trunk* [15].

Mód	**Popis**	**Zranitelný**
dynamic desireable	Port se aktivně snaží ustavit nový trunk link pomocí DTP protokolu. Pokud sousední přepínač nepodporuje trunking, je tento mód ekvivalentní access módu.	Ano
dynamic auto	Port se nesnaží aktivně vytvořit nový trunk link, ale pasivně naslouchá příchozím DTP rámcům a v případě zachycení požadavku je schopen trunk link vytvořit.	Ano
trunk	Trunking je v tomto módu vždy povolen nezávisle na sousedním přepínači.	Ano
access	Trunking je v tomto módu vždy zakázán nezávisle na sousedním přepínači.	Ne

Tabulka 2.1: Přehled módu přepínače a jejich popis.

Útočník se pak snaží přepínač přesvědčit, že jeho stanice je nový přepínač, který se k síti připojil. Zašle tedy požadavek protokolu *DTP* na sousední přepínač, čímž dojde k vytvoření dynamického trunk linku. Útočník pak zašle rámec označený tagem, který specifikuje VLAN síť oběti. Pokud by nedošlo k úspěšnému vytvoření trunk linku, rámec může být buď zahozen, nebo nezávisle na tagu přeposlán pouze do lokální VLAN sítě. V opačném případě switch důvěřuje tagu, který je uveden v hlavičce rámce a dojde k jeho přeposlání do VLAN oběti [15].

Oproti předchozímu útoku ze sekce 2.11.1 patří mezi výhody této metody především fakt, že spojení mezi obětí a útočníkem může být ustaveno obousměrně, jakoby neexistovala žádná bezpečnostní omezení. Hlavním problémem tohoto útoku je pak nemožnost jeho realizace v případě, že se port nachází v módu *access*. Vzhledem k tomu, že útok je

možné provést díky nesprávné konfiguraci portů na přepínači, nebyl doposud představen žádný přímý bezpečnostní mechanismus pro jeho eliminaci a veškerá zodpovědnost je tak na síťovém administrátorovi, který provádí konfiguraci přepínačů provádí. [15].

Podobně jako u předchozí metody nelze tento typ útoku detekovat s naprostou jistotou, protože v zachycené síťové komunikaci není možné rozlišit, zda žádosti protokolu *DTP* o dynamické vytvoření trunk linku jsou generovány legitimním síťovým zařízením, či útočníkem. V případě nalezení takové žádosti uvnitř komunikace je tedy vypsáno varovné hlášení, díky kterému může síťový administrátor na základě znalosti konfigurace sítě určit, zda se jedná o útok, či nikoliv.

2.12 Skenování sítě

Skenování sítě se obvykle přímo neklasifikuje jako síťový útok především z důvodu, že tento mechanismus nemusí být vždy využíván pro nekalé úmysly. Jedná se o proces identifikace aktivních zařízení v síti[17], jehož výsledkem je seznam všech jejích aktivních prvků včetně korespondujících IP adres. Velmi často jej využívají síťoví administrátoři za účelem bezpečnostní analýzy a hledání nežádoucích zařízení v síti, nicméně může být používán i útočníky pro hledání potenciálních obětí. Skenování obecně funguje na principu zasílání různých typů zpráv do sítě, na které aktivní zařízení v síti patřičně reagují. Z přijatých odpovědí je pak možné získat potřebné informace. Obvykle jsou tyto zprávy zasílány na konkrétní IP adresu, či na určitý rozsah IP adres. V následujících podkapitolách jsou uvedeny techniky užívané pro skenování lokálních *IPv4* a *IPv6* sítí. Metody skenování v obou typech sítí jsou poměrně odlišné, vzhledem k jejich rozdílným vlastnostem [19].

2.12.1 Skenování IPv4 sítě

Oproti *IPv6* sítím se *IPv4* sítě vyznačují výrazně nižším počtem stanic, které lze adresovat v rámci dané lokální sítě. Z tohoto důvodu je možné, aby skenování probíhalo sekvenčním zasíláním zpráv určitého typu na kompletní rozsah *IPv4* adres, které je možné v rámci lokální sítě přidělit. K tomu je obvykle využíván protokol *ARP*, jehož základní popis je uveden na začátku kapitoly 2.6.

Princip je tedy založen na sekvenčním zasílání *ARP* dotazů na všechny možné adresy, které je možné přidělit zařízením v rámci dané lokální sítě. Aktivní zařízení v síti pak na tento dotaz reagují zasláním *ARP reply*, čímž signalizují svojí aktivitu a zároveň poskytnou mapování mezi svojí *IP* a *MAC* adresou. Na základě těchto informací si pak útočník může vytvořit seznam potenciálních obětí, jež může podrobit dalšímu zkoumání.

Pro úspěšnou detekci skenování je zapotřebí znalost o počtu síťových zařízení připojených v dané lokální síti nebo celkový počet stanic, které je v ní možné adresovat. Vzhledem k tomu, že při skenování sítě pomocí *ARP* protokolu jsou zasílány dotazy na všechny přidělitelné IPv4 adresy v lokální síti, stačí detekovat zprávy typu *ARP request* a zjistit všechny unikátní *IPv4* adresy, které jsou dotazovány. Pokud jejich počet výrazně převyšuje množství připojených stanic v síti, či se blíží celkovému možnému počtu adresovatelných zařízení v lokální síti, je velice pravděpodobné, že se jedná o skenování sítě.

[17] Anglicky je tento proces označován termínem *Host Discovery*.

2.12.2 Skenování IPv6 sítě

Hlavním důvodem pro navržení protokolu *IPv6* bylo především nedostatečné množství adres poskytovaných protokolem IPv4. Adresový prostor *IPv6* tedy obsahuje oproti svému předchůdci enormní množství adres, které je možné použít pro adresování koncových zařízení. Principy zmíněné v sekci 2.12.1 nejsou tedy již z časových důvodů prakticky použitelné a je tedy nezbytné hledat jiné techniky pro skenování *IPv6* sítí [9].

První a nejpřímočařejší metodou je zaslání zprávy *Echo Request* protokolu *ICMPv6* do multicastové skupiny, jejímiž příjemci jsou všechny stanice v rámci dané lokální sítě (cílová *IPv6* adresa *ff02::1*). Aktivní uzly na tuto zprávu reagují zasláním *Echo Reply*, čímž dojde k získání potřebných informací o potenciálních obětech. Tato technika patří k těm nejefektivnějším, které lze při skenování využít, jelikož přímo využívá prostředky poskytované protokolem *ICMPv6*. Jeho nevýhodou je ovšem fakt, že systémy *Microsoft Windows* obvykle ignorují zprávy *Echo request* zaslané na multicastové adresy. Z tohoto důvodu je nutné pro skenování použít i další techniky [9].

Dalším možným způsobem je zaslání *ICMPv6* paketu, jenž má ve své hlavičce uvedený neznámý typ *ICMPv6* zprávy. Paket je opět směřován všem stanicím v rámci lokální sítě. Uzly neprodleně reagují zasláním zprávy *Parameter Problem*, čímž dají najevo svojí aktivitu v síti. Výhodou této techniky je především její úspěšnost i při skenování aktivních uzlů, které využívají systém *Microsoft Windows* [9].

Často používanou technikou je rovněž zaslání zprávy *Multicast Listener Query*, jež je součástí protokolu *Multicast Listener Discovery* (*MLD*). Tato zpráva je obvykle zasílána směrovačem vícesměrového vysílání, jejímž účelem je zjistit členy multicastové skupiny v rámci daného síťového segmentu. Tyto zprávy mohou být specifického charakteru (dotaz na členství v konkrétní skupině), či obecného charakteru (dotaz na členství ve všech skupinách) [6]. Paket tohoto typu je zaslán všem stanicím v rámci lokální sítě. Rovněž je v něm nastaveno pole *maximum response delay*[18] na hodnotu 0, čímž jsou hosté vynuceni zaslat odpověď ihned po přijetí žádosti, namísto čekaní na ostatní odpovědi z jejich multicastové skupiny. Aktivní uzly reagují zasláním zprávy *Multicast Listener Report*, čímž je odhalena jejich přítomnost [9].

Kromě výše uvedených metod existují i další, nicméně tyto tři patří k těm nejefektivnějším. Další možnosti pro skenování *IPv6* adresového prostoru, včetně jejich principů, je možné nalézt v *RFC 7707* [9]. Vzhledem k tomu, že některé způsoby nefungují na určité operační systémy, je vhodné tyto metody vzájemně kombinovat.

Detekovat všechny výše uvedené metody *IPv6* skenování ze zachycené síťové komunikace je poměrně přímočaré. Stačí vyhledat všechny zaslané pakety používané pro skenování *IPv6* sítě, vykazující výše uvedené vlastnosti. Pokud je nějaký z těchto paketů nalezen, je možné konstatovat pozitivní detekci skenování *IPv6* sítě.

2.13 Skenování portů

Po dokončení identifikace potenciálních obětí v síťovém segmentu, pomocí technik zmíněných v sekci 2.12, obvykle následuje proces zvaný skenování portů. Jedná se o metodu, která zahrnuje zasílání *TCP* segmentů a *UDP* datagramů na různé porty stanice oběti, za účelem identifikace, které z nich jsou otevřené. Tento proces je nezbytný pro určení služeb, které stanice oběti poskytuje a v důsledku i nalezení zranitelností, jež mohou potenciálnímu útočníkovi dovolit neautorizovaný přístup do cílového systému [17]. Důvodem je nejčastěji

[18] Toto pole určuje maximální možnou prodlevu před zasláním odpovědí na odpovídající dotaz.

chybná konfigurace těchto služeb, nebo využívání jejich verzí se známými bezpečnostními trhlinami [34]. Rovněž lze pomocí skenování zjistit i informace o používaném operačním systému, včetně jeho verze. Analogicky jako u skenování sítě je nutné zmínit, že se nemusí vždy jednat o nežádoucí techniku, jelikož bývá často používána síťovými administrátory za účelem zjištění bezpečnostních trhlin vedoucí ke zvýšení bezpečnosti systémů.

Porty jsou čísla z rozmezí 1–65 535[19], což znamená, že je nutné celkově provést 65 535 různých skenování pro kompletní prozkoumání otevřených portů na cílovém systému. Časová náročnost této operace může být u některých typů skenování poměrně vysoká. Přestože skenování celého rozsahu poskytuje nejpřesnější výsledek, jsou z časových důvodů často pouze skenovány tzv. *well-known* porty[20], které jsou vyhrazeny pro nejběžnější typy služeb. Pouze skenováním portů v tomto rozsahu je pak útočník schopen s vysokou pravděpodobností úspěchu odhalit požadované zranitelnosti [11].

Obecně je princip skenování založen na zasílání *TCP* segmentů a *UDP* datagramů na různé porty stanice oběti. Podle přijaté odpovědi je pak možné zjistit, v jakém stavu se port nachází. V následujících podkapitolách budou zmíněny konkrétní používané techniky, jejich principy a charakteristické vlastnosti.

Detekovat tento proces je možné analogicky, jako tomu bylo u útoku *SYN flood*. Pro každou metodu skenování je typická určitá posloupnost zaslaných paketů mezi útočníkem a obětí v případě, že skenovaný port je uzavřen. Ty jsou pro jednotlivé metody uvedeny v podkapitolách níže. Výskyt takové posloupnosti paketů nebývá v síti příliš typický a tudíž při detekci většího množství takových posloupností je možné konstatovat, že dochází ke skenování portů. Jejich počet je pak možné porovnat s určitým prahem, jehož hodnota se může odvíjet např. podle množství služeb, které stanice oběti poskytuje a podobně.

2.13.1 TCP SYN skenování

Tento typ skenování patří mezi nejzákladnější používané techniky. Často bývá nazýváno jako tzv. *half-open* skenování, protože nikdy nedojde k ustavení kompletního *TCP* spojení. Princip je založen na zaslání *SYN* paketu na konkrétní port stanice oběti. Pokud je v reakci na tuto zprávu přijat paket *SYN+ACK*, nebo pouze *ACK*, jedná se o otevřený port, v případě přijetí paketu *RST+ACK* pak o port uzavřený. Jestliže i přes opakované znovuzaslání nedojde k přijetí žádné odpovědi, nebo je přijata zpráva *ICMP Destination unreachable*, je port označen jako filtrovaný [13, 17].

Výhodou tohoto mechanismu je jeho rychlost a nenápadnost, díky čemuž je obtížně detekovatelný systémy firewall. Za nevýhodu je pak možné považovat fakt, že pro jeho realizaci jsou využívány tzv. *RAW* pakety, což vyžaduje administrátorská práva na stanici, ze které je skenování prováděno [13].

2.13.2 TCP connect skenování

Metoda *TCP connect* je založena na podobném principu jako předchozí typ skenování s tím rozdílem, že vždy dojde k ustavení kompletního *TCP* spojení. To je způsobeno tím, že pro navázání spojení je využito volání operačního systému *connect* namísto *RAW* paketů. Stav portu se pak určuje na základě návratové hodnoty funkce *connect* [13, 17].

Výhodou tohoto přístupu je především fakt, že pro jeho provedení nejsou vyžadována žádná speciální privilegia. Nevýhodou je pak jeho nižší rychlost a větší detekovatelnost díky

[19] Port číslo 0 je rezervován.

[20] Konkrétně se jedná o porty z rozmezí 1–1023.

ustavení kompletního spojení. Dále také menší možnost kontroly zasílaných paketů. Obecně platí, že pokud je možné provést *TCP SYN* skenování, jedná se obvykle o lepší variantu [13, 34].

2.13.3 UDP skenování

Při využití této techniky je na každý cílový port zaslán *UDP* paket, na který je reagováno zprávou *Port unreachable* protokolu *ICMP*, pokud je port uzavřený. Jestliže je přijata *ICMP* zpráva s jiným kódem, port je označen jako filtrovaný. Pokud služba odpoví zasláním *UDP* paketu, jedná se o otevřený port. V případě, že i po několikanásobném znovuzaslání paketu není ze strany oběti reagováno, znamená to, že port může být otevřen, či filtrován [13, 17].

Výsledky tohoto typu skenování mohou být poměrně nepřesné vzhledem k faktu, že *UDP* je nespojovaný protokol a nezaručuje spolehlivé doručení. Další nevýhodou je taktéž velká časová náročnost jeho provedení. Z těchto důvodů ovšem *UDP* skenování bezpečnostní auditoři často opomíjí, což může být považováno za jeho výhodu [17].

2.13.4 TCP NULL, FIN, Xmas a Maimon skenování

Všechny tyto čtyři typy skenování jsou založené na stejném principu, rozdíl je pouze v nastavení různých *TCP* flagů v hlavičce paketu, který je zasílán na stanici oběti. V případě *NULL* skenování nejsou v hlavičce nastaveny žádné bity, *FIN* skenování má nastaveno pouze *TCP FIN* bit, technika *Xmas* má v hlavičce nastaveny bity *FIN*, *PSH* a *URG* a *Maimon* využívá nastavení bitu *FIN* společně s *ACK* [17].

Pro rozlišení stavu portů je v těchto případech využíváno mezer v definici *TCP* protokolu, kterou je možné nalézt v *RFC 793* [24]. Pokud je cílový port uzavřen a dojde k zaslání některého z výše uvedených paketů, je reagováno paketem *RST+ACK* v případě skenováni typu *NULL*, *Xmas* nebo *FIN*. Při skenování typu *Maimon* je reagováno paketem *RST*. V případě žádné odpovědi je port označen za uzavřený, či filtrovaný. Port je rovněž označen za filtrovaný při přijetí zprávy *ICMP Destination unreachable* [17].

Výhodou těchto přístupů je jejich nenápadnost, která jim umožňuje projít některými typy firewallů a routerů filtrujících pakety. Mezi nevýhody lze zařadit nefunkčnost těchto mechanismů na různých systémech (především *Microsoft Windows*) z důvodu volnější implementace pravidel uvedených v *RFC 793* [24].

2.13.5 TCP ACK skenování

Technika *TCP ACK* je využívána především pro identifikaci, zda je na stanici oběti přítomen stavový, či nestavový firewall a k zjištění pravidel, která jsou v něm nastavená. K tomuto účelu je využíván segment, který má v *TCP* hlavičce nastavený pouze *ACK* bit. Porty, které jsou uzavřené, či otevřené, odpoví paketem *RST*, což značí že nedochází k jejich filtrování. Pokud není zaslána žádná odpověď, nebo zpráva protokolu *ICMP Destination unreachable*, je port označen za filtrovaný [17].

2.13.6 TCP Window skenování

TCP Window skenování je založeno na stejném principu jako předchozí technika, nicméně na rozdíl od ní je schopno rozlišit, zda jsou cílové porty otevřené, či uzavřené. Využívá při tom faktu, že některé systémy pro otevřené porty v zaslaném *RST* paketu používají kladnou hodnotu pro velikost okna (angl. *window size*), zatímco pro ty uzavřené je tato

hodnota nastavena na nulu. Je nezbytné zmínit, že tato technika je poměrně nespolehlivá, protože některé systémy mohou pro oba typy portů nastavovat velikost okna na nulovou hodnotu, čímž označí všechny porty za uzavřené [17].

Kapitola 3

Návrh aplikace

V předchozí kapitole byl uveden výčet síťových útoků, jejichž automatizovanou detekci ze zachycené síťové komunikace by měl výsledný nástroj poskytovat. Byly rozebrány jejich principy a zranitelnosti, kterých útoky využívají, včetně způsobu jejich rozpoznání na úrovni zachycené síťové komunikace. Na základě těchto znalostí je zapotřebí navrhnout podobu aplikace, která bude schopna jednotlivé útoky detekovat. Hlavní požadavek, jemuž je při návrhu zapotřebí věnovat zřetel, spočívá v možnosti snadného přidávání nových typů útoků, jež bude schopen nástroj detekovat, aniž by bylo zapotřebí provádět výrazné změny v implementaci. Z tohoto důvodu je nevhodné, aby proces rozpoznávání jednotlivých útoků byl přímo implementován v kódu aplikace.

Existuje zajisté větší množství přístupů, jak výše uvedený požadavek realizovat, nicméně za nejvhodnější řešení byl zvolen deklarativní zápis útoku popisující způsob jeho detekce. Je tedy nutné navrhnout komplexní způsob zápisu, který by poskytoval prostředky pro popis detekce širší škály síťových útoků a který by byl snadno strojově zpracovatelný. Samotná aplikace pak bude fungovat jako interpret, jehož vstupem bude soubor se zachycenou síťovou komunikací a textové soubory obsahující deklarativní zápisy jednotlivých útoků, popisující způsoby detekce těchto útoků.

Proces zpracování bude spočívat v převedení zachycené komunikace pomocí nástroje *tshark* do formátu PDML (založeném na XML), ve kterém se pak pomocí výše zmíněných deklarativních zápisů bude ověřovat přítomnost síťových útoků. Za tímto účelem byl tedy navrhnut formát pro jejich deklarativní zápis. Popisem syntaxe, sémantiky a způsobem interpretace tohoto formátu se zabývá podkapitola 3.1. V rámci zadání byly taktéž vzneseny požadavky na vstupní soubory se zachycenou komunikací. Jejich popis je uveden v podkapitole 3.2.

Pro zpracování zachycené síťové komunikace ve formátu PDML je možné využít některou z dostupných knihoven pro automatizované parsování XML dokumentů. Při výběru knihovny je nicméně nezbytné brát ohled na rychlost jejich zpracování vzhledem k vysoké velikosti souborů formátu PDML[1]. Jako nejvhodnější se tedy jeví použití knihovny *cElementTree*[2] pro jazyk *Python 2.7*, ve kterém bude výsledný nástroj implementován. Tato knihovna je napsána v čistém jazyce *C* a z tohoto důvodu ve srovnávacích testech[2] dosahuje nejlepších výsledků. Pro navigaci v XML dokumentu pak bude použit jazyk *XPath*.

[1] Tato velikost bývá občas až v rámci gigabytů.

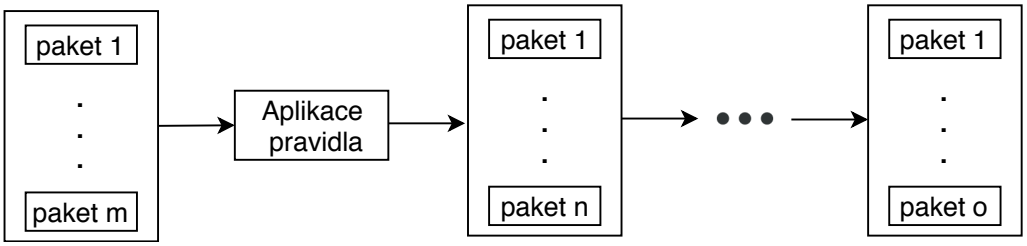
[2] Viz. http://effbot.org/zone/celementtree.htm

3.1 Formát pro deklaraci útoků

Prostředkem pro deklarativní popis síťových útoků byl zvolen serializační formát *YAML*. Mezi jeho vlastnosti patří především dobrá čitelnost jak člověkem, tak i strojem, která je v tomto případě klíčová. V následujícím textu bude uvedena struktura popisu, již je zapotřebí dodržet, aby mohl být útok korektně strojově zpracován a detekován.

Útok je popsán strukturou, jejíž první atribut má název **name**, který obsahuje textový řetězec reprezentující jméno útoku. Za ním následuje **scope**, jehož hodnotou musí být jeden z řetězců *atomic*, *stream*, nebo *group*. Hodnota scope určuje, zda nás při hledání útoku v dané síťové komunikaci zajímají konkrétní pakety, u kterých není zapotřebí znalost kontextu[3], ve kterém byly zaslány (*atomic*), obousměrná komunikace mezi dvěma síťovými entitami, což znamená jednotlivé *TCP*, nebo *UDP* streamy (*stream*), nebo sekvence paketů, které jsou sice zasílány mezi dvěma entitami v síti, ale nelze je navzájem sdružit[4] do obousměrné komunikace obdobně jako v předchozím případě (*group*). Pro každou tuto hodnotu je aplikován rozdílný postup při zpracování síťové komunikace a rovněž vyžadují i specifické atributy. Průběh tohoto zpracování, včetně popisu specifických atributů je tedy vyčleněn ve speciálních kapitolách 3.1.2, 3.1.3 a 3.1.4. Následující odstavce budou věnovány popisu dalších atributů společných pro všechny tři typy.

Po atributu scope následuje vlastnost s názvem **properties**. Jedná se o kolekci struktur, pomocí níž dochází k filtrování paketů, které jsou objektem zájmu při zkoumání útoku. V případě, že tento atribut není uveden, jsou při hledání útoku uvažovány všechny pakety. Je-li to ovšem možné, doporučuje se, aby byl tento atribut přítomen, protože tím dojde k výraznému zrychlení při detekci útoku. Jednotlivé struktury v kolekci pak představují samotná filtrovací pravidla. Formát jejich zápisu je uveden v podkapitole 3.1.1. Filtrovací proces probíhá tak, že dojde k aplikaci prvního pravidla na množinu všech paketů. Z těch jsou vybrány pouze ty, které danému pravidlu vyhovují. Na takto získanou množinu paketů jsou pak postupně aplikována další pravidla, dokud není aplikováno pravidlo poslední. Výsledná množina obsahuje tedy pouze pakety, které vyhovují všem uvedeným pravidlům. Proces filtrace je graficky znázorněn na obrázku 3.1.



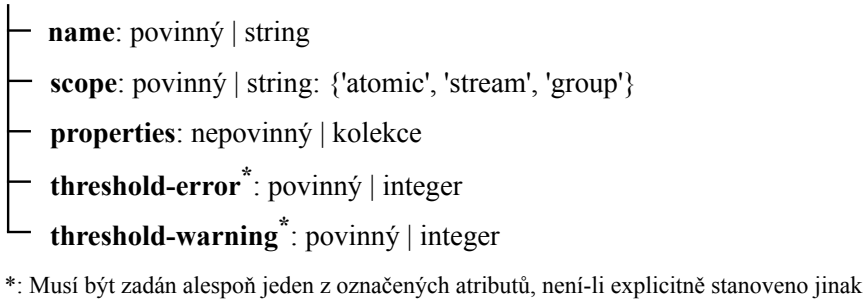
Obrázek 3.1: Grafické znázornění filtrování paketů. Pro čísla m, n, o platí $m \geq n \geq o$.

Posledními společnými atributy jsou **threshold-error** a **threshold-warning**, které očekávají jako svoji hodnotu nezáporné celé číslo. Tyto hodnoty určují minimální množství pozitivních nálezů po dokončení procesu detekce útoku, které je třeba detekovat, aby bylo vypsáno hlášení o nalezení útoku, případně varování o potenciálním útoku. Jejich bližší

[3] Tzn. není nutné pro detekci útoku zkoumat celý *TCP*, nebo *UDP* stream, jehož je paket součástí, ale útok lze detekovat z konkrétního paketu, aniž by bylo zapotřebí analyzovat pakety související se zkoumaným paketem.

[4] Pomocí pětice zdrojová IP adresa, cílová IP adresa, zdrojový port, cílový port, protokol TCP/UDP.

význam pro každou z hodnot atributu *scope* je uveden v kapitolách 3.1.2, 3.1.3 a 3.1.4. V případě uvedení obou atributů najednou je počet pozitivních nálezů porovnán prvně s hodnotou *threshold-error*. Pokud je tento počet větší nebo roven uvedené hodnotě prahu, dojde k vypsání hlášení o nalezení útoku. V opačném případě proběhne stejné porovnání s hodnotou *threshold-warning* a vypsání varování o potenciálním výskytu útoku, pokud je podmínka splněna. Jestliže nedojde ke splnění ani této podmínky, útok není detekován. Shrnutí výše uvedených atributů, včetně jejich datových typů a povinnosti jejich zadání, je uvedeno na obrázku 3.2. Podobná shrnutí atributů budou obsažena i dále v textu. Jejich zápis bude uveden vždy ve formátu „**název atributu**: povinnost zadání | datový typ“.



Obrázek 3.2: Shrnutí atributů společných pro všechny typy útoků.

3.1.1 Zápis filtrovacích pravidel

Před samotným popisem struktury pro zápis filtrovacích pravidel je nutné zmínit, že každé pole paketu je ve formátu PDML reprezentováno jedním XML uzlem. Název pole není ovšem určen názvem XML uzlu, ale je specifikován pomocí atributu *name*[5] daného uzlu.

Prvním atributem struktury pro popis filtrovacích pravidel je **field-name**. Textový řetězec jemu přiřazený určuje hodnotu XML atributu *name* libovolného[6] uzlu v PDML souboru se zachycenou síťovou komunikací. Jinými slovy hodnota atributu *field-name* určuje název pole v paketu, jež nás bude při filtrování zajímat. Dalším atributem je **valid**, který může nabývat hodnot *true* a *false*. Filtrovací pravidlo sestávající pouze z těchto dvou položek znamená, že budou vyfiltrovány všechny pakety, ve kterých se pole s daným názvem vyskytuje (pokud je atribut *valid* je nastaven na *true*), nebo naopak všechny pakety, ve kterých se specifikované pole nevyskytuje (pokud je atribut *valid* nastaven na *false*). V příkladu č. 1 je uvedeno pravidlo, které vyfiltruje všechny pakety obsahující pole s názvem *arp*, tedy všechny zprávy protokolu ARP zachycené v předložené síťové komunikaci. Pokud by atribut *valid* byl nastaven na hodnotu *false*, výsledkem aplikace pravidla by byly všechny pakety kromě výše zmíněných *arp* paketů.

```
field-name: arp
valid: true
```

Příklad 1: Pravidlo pro filtrování ARP paketů.

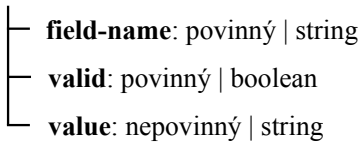
[5] Jako příklad je možné uvést, že zdrojovou IP adresu paketu lze nalézt v XML uzlu s názvem *field* s hodnotou atributu *name*="ip.src".

[6] V tomto případě slovo „libovolného“ značí jakýkoliv uzel nehledě na úroveň jeho zanoření v PDML souboru.

Dále je možné filtrovací pravidlo více konkretizovat přidáním dalšího atributu **value**. Tato položka se použije v případě, kdy nás u konkrétního pole paketu, specifikovaného pomocí položky *field-name*, zajímá jeho hodnota. Položka *value* očekává textový řetězec, jehož hodnota odpovídá hodnotě XML atributu s názvem *show* u specifikovaného pole v PDML souboru. Příklad č. 2 uvádí pravidlo, které prvně nalezne pole s názvem *ip.src* a v něm hledá XML atribut s názvem *show*, jehož hodnota odpovídá řetězci *192.168.0.1*. Toto konkrétní filtrovací pravidlo tedy vyfiltruje všechny pakety, jejichž zdrojová IP adresa má výše uvedenou hodnotu. Pokud by atribut *valid* byl nastaven na hodnotu *false*, vyfiltrovaly by se všechny pakety, které pole s názvem *ip.src* sice obsahují, ale jeho hodnota je rozdílná od hodnoty specifikované atributem *value*. Jestliže paket v tomto případě pole *ip.src* neobsahuje vůbec, nebude pro něj toto pravidlo platit a bude odfiltrován. Vlastnosti všech výše uvedených atributů jsou uvedeny na obrázku 3.3.

```
field-name: ip.src
value: 192.168.0.1
valid: true
```

Příklad 2: Pravidlo pro filtrování paketů na základě zadané zdrojové IP adresy.



Obrázek 3.3: Shrnutí atributů pro zápis filtrovacího pravidla.

3.1.2 Zpracování útoku typu atomic

Typ útoku s označením *atomic*, určeným pomocí atributu *scope*, se vyznačuje tím, že pro jeho detekci postačuje procházet jednotlivé pakety v zachycené síťové komunikaci a u každého z nich ověřovat uživatelem specifikované podmínky, jež daný útok identifikují. Pro tento typ útoku je podstatné, že pro jeho odhalení není důležité, v jakém síťovém toku se zkoumané pakety vyskytují. Z toho vyplývá, že pro korektní detekci útoku není zapotřebí zjišťovat, jaké další pakety jsou obsaženy ve stejném síťovém toku, v němž se vyskytuje aktuálně zkoumaný paket. Obecně platí, že pro úspěšnou detekci útoku může být zapotřebí více paketů, nicméně tyto pakety mohou být součástí různých síťových toků.

Kromě společných atributů, uvedených v kapitole 3.1, má tento typ útoku navíc pouze jeden **povinný** atribut s názvem **detection-conditions**. Jedná se o strukturu obsahující kolekci, ve které jsou uvedeny jednotlivé detekční podmínky. Tento atribut je zde stěžejní, protože obsahuje popis detekčních podmínek, které slouží k identifikaci daného síťového útoku. Podrobný popis dostupných podmínek, jejich sémantiky a způsobu zápisu je možné nalézt v kapitole 3.1.5.

Při samotné detekci útoku je pak v prvním kroku proveden proces filtrace pomocí podmínek uvedených v kolekci označené atributem *properties*. Tím dojde k odstranění zbytečných paketů, které zpomalují proces detekce. Následně je výsledek filtrování sekvenčně procházen paket po paketu a dochází k postupnému ověřování jednotlivých podmínek, uvedených

v sekci *detection-conditions*. V případě, že vlastnosti paketu vyhovují těmto podmínkám, je inkrementováno počítadlo pozitivních detekcí útoku. Pokud hodnota tohoto počítadla dosáhne hodnoty uvedených prahů *threshold-warning* nebo *threshold-error*, dojde k vypsání varovného či chybového hlášení a detekce je ukončena. Jestliže pak nebylo dosaženo ani jedné z hodnot, je vypsána zpráva o negativní detekci. Konkrétní příklad kompletního zápisu tohoto typu útoku je demonstrován na útoku *LAND* a je možné jej najít v příloze B.

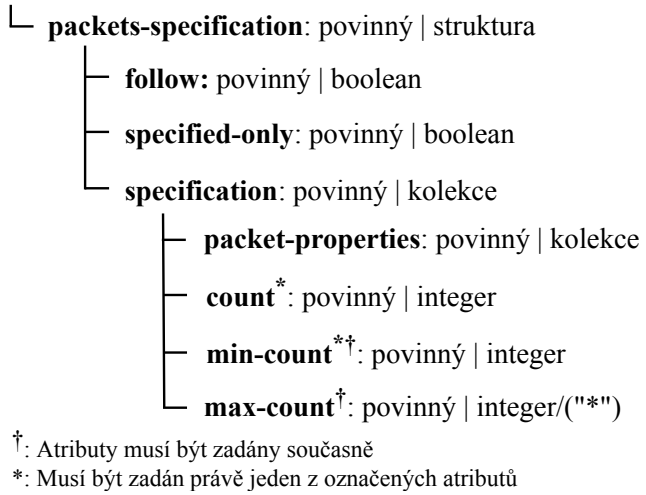
3.1.3 Zpracování útoku typu stream

Útok typu *stream* je specifický tím, že síťové útoky jsou v tomto typu útoku vyhledávány v obousměrné komunikaci mezi dvěma síťovými entitami, identifikovanými prostřednictvím IP adres, na konkrétních portech pomocí transportního protokolu *TCP* či *UDP*. Pakety jsou tedy před samotným vyhledáváním síťových útoků rozčleněny do odpovídajících *TCP*, či *UDP* streamů na základě indexu streamu, který do paketů automaticky doplňuje nástroj *tshark*. Tyto streamy jsou následně skenovány za účelem nalezení síťového útoku. Detekce útoku probíhá na základě výskytu uživatelem specifikovaných paketů v jednotlivých streamech.

Oproti společným atributům je zde navíc atribut **packets-specification**, který slouží pro specifikaci jednotlivých paketů, které se v daném toku vyskytují a jejichž prezence indikuje síťový útok. Její hodnotou je struktura obsahující tři atributy. První dva mají název **follow** a **specified-only**. Oba mohou nabývat pouze logických hodnot *true* a *false* a jejich význam bude uveden dále v textu. Třetí atribut má pak název **specification**. Jedná se o kolekci, jejíž každá jedna položka popisuje vlastnosti jednoho konkrétního typu paketu a počet jeho výskytů ve streamu. Struktura pro jeho popis se pak skládá z atributu **packet-properties**, což je kolekce obsahující soubor pravidel specifikující vlastnosti paketu. Sémantika tohoto atributu, způsob zápisu jednotlivých pravidel a postup vyhodnocení je naprosto identický jako u položky *properties*, která je uvedena v kapitole 3.1. Dále obsahuje atribut **count**, který pro každý paket určuje počet jeho výskytů v toku. Alternativně je možné ho nahradit dvojicí **min-count** a **max-count**, kterou je možné specifikovat interval určující rozsah počtu výskytů. Pokud hodnota atributu *max-count* obsahuje symbol "*"(včetně uvozovek), je horní hranice intervalu nespecifikována. Výše uvedené atributy včetně jejich datových typů, povinnosti zadání a vzájemné hierarchie jsou shrnuty na obrázku 3.4.

Detekce útoku je zahájena filtrováním paketů na základě jejich společných vlastností, které jsou specifikované pomocí atributu *properties*. Všechny takto získané pakety jsou následně rozděleny do streamů na základě indexu *TCP*, nebo *UDP* streamu, který k paketům automaticky doplňuje nástroj *tshark*. V každém streamu je následně kontrolováno, zda se v něm skutečně vyskytují pakety, které jsou specifikované v kolekci *specification*, včetně jejich odpovídajícího počtu. Pokud je atribut *follow* nastaven na hodnotu *true*, je navíc kontrolováno, zda specifikované pakety uvedené v kolekci *specification* následují v daném streamu bezprostředně za sebou ve stejném pořadí, jako jsou uvedeny v kolekci. V případě nastavení atributu *specified-only* na hodnotu *true* musí platit, že se ve streamu nevyskytují jiné pakety, než ty specifikované v kolekci *specification*. Pokud jsou oba parametry *follow* a *specified-only* nastaveny na hodnotu *false*, ověřuje se pouze výskyt specifikovaných paketů ve streamu a zda odpovídá jejich počet. Neřeší se už jejich pořadí ani výskyt jiných paketů v síťovém toku, než jsou ty specifikované.

Pokud daný stream splňuje všechny výše uvedené podmínky, je označen za pozitivní nález a je inkrementováno příslušné počítadlo, které je po procesu prohledávání všech streamů porovnáno s hodnotami prahů *threshold-warning* a *threshold-error*. Pokud počítadlo



Obrázek 3.4: Shrnutí atributů specifických pro útok typu *stream*.

tyto hodnoty přesáhne, dojde k vypsání varovného, či chybového hlášení o detekci síťového útoku. Konkrétní příklad kompletního zápisu tohoto typu útoku je demonstrován na skenování portů metodou *TCP SYN* a je možné jej najít v příloze C.

Příklad č. 3 uvádí zápis útoku, který je možné identifikovat pomocí streamu, jenž obsahuje paket, jehož hlavička má nastaveno pole *tcp.flags* na hodnotu rovnou dvěma (hexadecimálně), což značí paket s nastaveným příznakem *SYN*. Atribut *follow* v tomto případě nemá žádný význam, protože kolekce specification obsahuje pouze jeden paket. Položka *specified-only* nastavená na *false* pak značí, že kromě specifikovaného paketu se může v prohledávaném streamu vyskytovat libovolný počet jiných paketů.

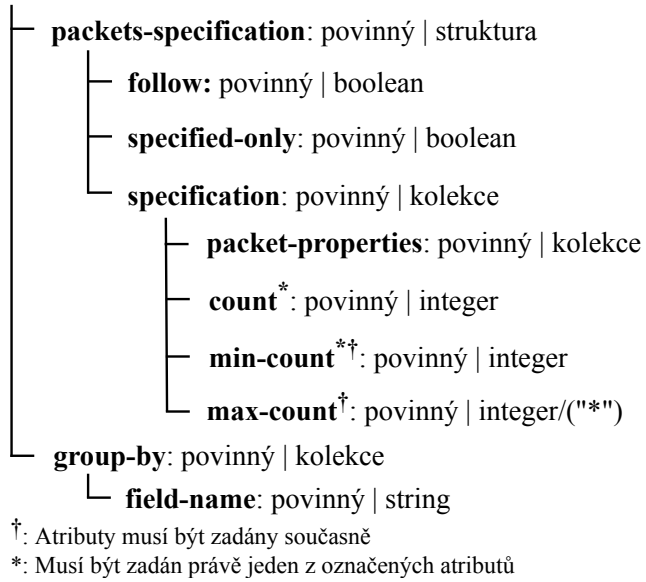
```
packets-specification:
  follow: false
  specified-only: false
  specification:
    - count: 1
      packet-properties:
        - field-name: tcp.flags
          attribute: value
          value: '0x00000002' # SYN paket
```

Příklad 3: Zápis toku obsahující jeden *SYN* paket protokolu *TCP*.

3.1.4 Zpracování útoku typu group

Útok typu *group* je zobecněním útoku typu *stream*. Obdobně jako v tomto typu útoku jsou útoky vyhledávány v určité skupině paketů, nicméně nedochází zde ke třídění paketů dle jejich příslušnosti do *TCP*, či *UDP* streamu. Uživatel v tomto případě musí specifikovat seznam polí, na základě jejichž společné hodnoty jsou pakety umístěny do identické skupiny. Ve vytvořených skupinách jsou pak hledány specifikované pakety, jejichž výskyt signalizuje přítomnost útoku.

Stejně jako u předchozího typu útoku i zde jsou přítomny **povinné** atributy **packets-specification**, **follow** a **specified-only**, jejichž zápis i sémantika je ekvivalentní stejnojmenným vlastnostem popsaným v kapitole 3.1.3. Poslední atribut nese jméno **group-by**. Jeho prostřednictvím jsou určeny názvy polí, která se mají hledat v jednotlivých paketech a na základě jejichž identických hodnot mají být pakety sloučeny do stejných skupin. Jedná se tedy o kolekci struktur, která se skládá pouze z jednoho atributu s názvem **field-name**, jenž obsahuje textový řetězec s názvem pole. Výše uvedené atributy včetně jejich datových typů, povinnosti zadání a vzájemné hierarchie jsou shrnuty na obrázku 3.5.



Obrázek 3.5: Shrnutí atributů specifických pro útok typu *group*.

Proces detekce je prakticky identický s postupem uvedeným na konci kapitoly 3.1.3. Jediný rozdíl nastává při seskupování paketů, během kterého dochází k sekvenčnímu procházení kolekce *group-by* od její první položky. Na základě položek v ní uvedených je vždy zkontrolováno, zda se v aktuálně prohledávaném paketu nachází pole se specifikovaným názvem. V kladném případě je paket přiřazen do skupiny identifikované hodnotou tohoto pole, v opačném se pak pokračuje další položkou v kolekci. Pokud se v paketu nenachází žádné z polí specifikované položkami v kolekci *group-by*, je takový paket ignorován. Pak už se ve skupinách vyhledávají jednotlivé specifikované pakety a kontroluje se, zda splňují vlastnosti určené atributy *follow* a *specified-only* (postup je tedy dále identický jako u útoku typu *stream*). Konkrétní příklad kompletního zápisu tohoto typu útoku je demonstrován na útoku *Duplicate address detection* a je možné jej najít v příloze D.

Příklad č. 4 uvádí zápis sekce *group-by*, která sloučí pakety do skupin na základě hodnot polí s názvem specifikovaným atributem *field-name*.

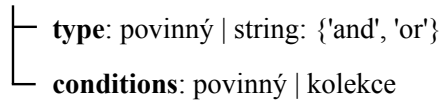
```
group-by:
  - field-name: icmpv6.nd.ns.target_address
  - field-name: icmpv6.nd.na.target_address
```

Příklad 4: Zápis sekce *group-by* specifikující pole pro sloučení paketů do skupin.

3.1.5 Zápis podmínek pro detekci

Obsah této podkapitoly je věnován důkladnému popisu různých typů podmínek, které je možné využít pro identifikaci paketů vykazujících známky síťového útoku. Jejich použití je možné pouze při zpracování útoků typu *atomic*, který je definován pomocí atributu *scope*. Důraz bude kladen především na vysvětlení sémantiky těchto pravidel, způsobu zápisu a postupu při jejich vyhodnocení.

Podmínky jsou součástí sekce *detection-conditions* a jsou popsány strukturou o dvou položkách. První z nich je atribut **type**, kterému mohou být přiřazeny pouze dvě řetězcové hodnoty *or*, nebo *and*. Ty určují, zda v případě uvedení většího množství podmínek je zapotřebí, aby platila alespoň jedna podmínka (*or*), nebo zda musí platit všechny uvedené (*and*). Pokud je uvedena pouze jedna podmínka, měl byt být tento atribut nastaven na hodnotu *and*. Druhým atributem je pak kolekce s názvem **conditions**, ve které jsou již uvedeny konkrétní podmínky, pomocí nichž dochází k detekci útoku. Výše uvedené atributy včetně jejich datových typů a povinnosti zadání jsou shrnuty na obrázku 3.6. Každá podmínka je pak popsána strukturou, v níž je pro všechny podmínky společný pouze atribut **condition-type**, který určuje konkrétní typ podmínky. Ostatní atributy této struktury jsou podmíněny právě tímto typem.



Obrázek 3.6: Shrnutí atributů specifických pro sekci popisující detekční podmínky.

Pokud je atribut *type* nastaven na hodnotu *or*, je nezbytné, aby atributy *threshold-error* a *threshold-warning* byly přesunuty z nejvyšší úrovně deklarativního popisu přímo na nejvyšší úroveň struktury popisující podmínku, aby bylo možné specifikovat různé prahy pro každou z podmínek. Vyhodnocení útoku pro hodnotu *or* atributu *type* probíhá tak, že se nejdříve vyfiltrují vstupní pakety pomocí filtrovacích pravidel uvedených v sekci *properties*. Následně jsou sekvenčně procházeny jednotlivé podmínky v kolekci *conditions*, které jsou nad množinou vyfiltrovaných paketů postupně vyhodnocovány. Počet pozitivních detekcí je pak porovnán s uvedenými prahy u každé podmínky a pokud je dosaženo alespoň hodnoty atributu *treshold-warning*, dojde k vypsání příslušného hlášení a ukončení detekce. Při nalezení první platné podmínky uvedené v kolekci *conditions* je vyhodnocování podmínek ukončeno a další podmínky již vyhodnocovány nejsou. Existují i případy, kdy se hodnoty prahů u podmínek neuvádí. Tyto případy jsou explicitně zmíněny v popisu jednotlivých podmínek. V takovém případě je výstup podmínky pouze hodnota *true* (splněna), nebo *false* (nesplněna) a nikoliv seznam paketů, pro které podmínka platí.

V případě hodnoty *and* jsou pak hodnoty prahů uvedeny na nejvyšší úrovni deklarativního popisu. Vyhodnocení opět začíná vyfiltrováním paketů uvedených v kolekci *properties*. Následně je určen seznam všech paketů, pro které platí všechny podmínky uvedené v seznamu. Počet pozitivních detekcí je pak porovnán s uvedenými prahy a je vypsáno příslušné hlášení o detekci či nenalezení útoku. Pokud je v kolekci podmínek uvedena taková podmínka, která vrací pouze hodnoty *true* (splněna), nebo *false* (nesplněna) (tzn. podmínka co nevyžaduje prahy), je na základě jejího splnění či nesplnění pouze určeno, zda se má ve vyhodnocování dalších podmínek pokračovat, nebo zda vyhodnocování zastavit. Pokud žádná z podmínek v kolekci *conditions* nevyžaduje prahy, není zapotřebí je na nejvyšší

úrovni deklarativního popisu definovat a v takovém případě dojde jen k ověření, zda jsou všechny podmínky splněny a nedochází tak k porovnání s prahy. V následujících podkapitolách jsou uvedeny všechny čtyři typy podmínek, které nástroj podporuje. Tyto podmínky lze libovolně kombinovat, pokud to v daném případě dává smysl.

Podmínka typu field-value

Tato podmínka je specifikována atributem *condition-type* s hodnotou *field-value*. Jejím prostřednictvím lze určit výskyt konkrétního pole v paketu, či přímo jeho hodnotu, které potvrzují přítomnost síťového útoku. Pokud je tedy specifikované pole v paketu nalezeno (a případně jestli souhlasí jeho hodnota, pokud je zadána), je paket označen za pozitivní detekci. Jméno pole je ve struktuře popisující podmínku určeno pomocí položky s názvem **field-name**. Pokud se ve struktuře už jiný atribut nevyskytuje, je tato podmínka při zpracování paketu vyhodnocena jako pravdivá, jestliže je v paketu obsaženo pole s tímto názvem. V souboru se zachycenou komunikací ve formátu PDML je tento název určen pomocí atributu *name* uvedeného u každého XML uzlu reprezentujícího pole paketu. Zápis konkrétní podmínky demonstruje příklad č. 5, který označí za pozitivní nález všechny pakety obsahující pole s názvem *ipv6.opt.unknown*. Počet pozitivních nálezů se pak porovná s hodnotami prahů uvedenými v zápisu útoku a vypíše se příslušné hlášení.

```
condition-type: field-value
field-name: ipv6.opt.unknown
```

Příklad 5: Podmínka detekující pakety, které obsahující pole s názvem *ipv6.opt.unknown*.

Tuto podmínku lze navíc rozšířit dvěma dalšími atributy, pomocí nichž lze blíže popsat hodnotu zkoumaného pole. Jejich přítomnost má smysl jen v případě, pokud jsou oba zadány společně. Atribut **value-type** může obsahovat pouze dvě řetězcové hodnoty *abstract* a *specific*, na základě kterých je určen obsah atributu **value**.

Pokud položka *value-type* nabývá hodnoty *specific*, v položce *value* se očekává konkrétní hodnota XML atributu *show* uvedeného v PDML souboru u pole s názvem specifikovaným atributem *field-name*. Při zpracování jsou pak sekvenčně procházeny všechny pakety a pokud v jejich těle dojde k nalezení specifikovaného pole s hodnotou odpovídající atributu *value*, je takový paket klasifikován jako pozitivní detekce. V příkladu č. 6 je uveden zápis pravidla, které za pozitivní nález označí všechny pakety ve vstupním PDML souboru, jenž obsahují pole *icmp.type* s hodnotou 5 (pakety *ICMP Redirect*). Počet pozitivních nálezů se pak porovná s hodnotami prahů uvedenými v zápisu útoku a vypíše se příslušné hlášení.

```
condition-type: field-value
value-type: specific
field-name: icmp.type
value: '5'
```

Příklad 6: Podmínka detekující pakety ICMP Redirect.

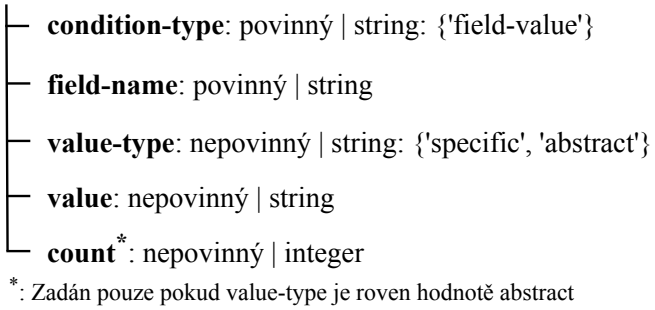
V případě, že atribut *value-type* obsahuje hodnotu *abstract*, je pak v položce *value* očekávána řetězcová hodnota *different* nebo *same*. V případě *different* jsou postupně procházeny všechny pakety v zachycené síťové komunikaci a dochází k zaznamenávání všech unikátních hodnot XML atributu *show* v PDML souboru u polí, jejichž název je specifikován atributem

field-name. Pokud počet těchto unikátních hodnot přesáhne celočíselnou hodnotu atributu **count**, je tato podmínka vyhodnocena jako pravdivá. Pokud atribut *value* odpovídá hodnotě *same*, jsou postupně procházeny jednotlivé pakety a u každé hodnoty specifikovaného pole je poznamenáno, v kolika dalších paketech se tato konkrétní hodnota vyskytla. Zjistí se tedy maximální počet paketů, které sdílí stejnou hodnotu daného pole. Následně dojde opět k porovnání s atributem *count* a vyhodnocení, zda je podmínka splněna či nikoliv. Příklad č. 7 demonstruje podmínku, která určí počet různých hodnot uvedených v atributu *show* pole s názvem *arp.dst.proto_ipv4*. V tomto případě se jedná o pole, které obsahuje dotazovanou *IPv4* adresu ve zprávě *ARP request*. Pomocí této podmínky lze po porovnání s atributem *count* identifikovat proces skenování sítě prostřednictvím protokolu *ARP*.

```
condition-type: field-value
value-type: abstract
field-name: arp.dst.proto_ipv4
value: different
count: 200
```

Příklad 7: Podmínka detekující proces skenování IPv4 sítě.

Pokud je hodnota atributu *value-type* nastavena na hodnotu *abstract*, platí zde výjimka, že konkrétně pro tuto podmínku není zapotřebí definovat ani jeden z prahů *threshold-error* nebo *threshold-warning*. Je to z důvodu, že tato podmínka se netýká konkrétně jednoho paketu, ale k jejímu vyhodnocení je zapotřebí více různých paketů a výsledkem po porovnání s atributem *count* je logická hodnota *true* nebo *false*. Nedává tedy smysl definovat prahy z toho důvodu, že v případě této podmínky nelze hovořit o konkrétním počtu pozitivních detekcí. Výše uvedené atributy včetně jejich datových typů, povinnosti zadání a vzájemné hierarchie jsou shrnuty na obrázku 3.7.



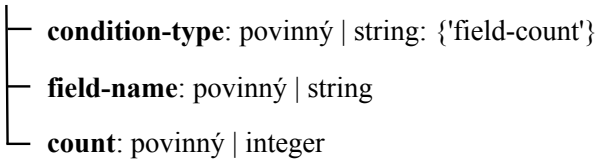
Obrázek 3.7: Shrnutí atributů specifických podmínku typu *field-value*.

Podmínka typu field-count

Podmínku typu *field-count* je možné využít při detekci útoku, který se vyznačuje opakováním určitého pole uvnitř těla paketu[7]. Zápis této podmínky obsahuje tři atributy, přičemž první z nich nese název **condition-type**. Obsahuje vždy hodnotu *field-count*. Druhý atribut **field-name** určuje název pole v PDML souboru se zachycenou síťovou komunikací, jehož

[7]Příkladem může být útok *Double tagging*, který se vyznačuje opakováním pole obsahující hodnotou tagu.

počet výskytů je zapotřebí ověřit. Poslední atribut **count** obsahuje celočíselnou hodnotu určující, kolikrát se pole specifikované pomocí atributu *field-name* musí v paketu vyskytovat, aby byl paket označen za pozitivní detekci. Výše uvedené atributy včetně jejich datových typů, povinnosti zadání a vzájemné hierarchie jsou shrnuty na obrázku 3.8.



Obrázek 3.8: Shrnutí atributů specifických podmínku typu *field-count*.

Při zpracování podmínky dochází k porovnání získaného počtu specifikovaných polí v paketu s hodnotou atributu *count* a pokud se počet rovná hodnotě tohoto atributu, je paket označen za pozitivní detekci. Celkový počet pozitivních detekcí v zachycené komunikaci se následně porovná s hodnotami uvedených prahů a je případně vypsáno patřičné hlášení. Příklad č. 8 uvádí zápis podmínky, která v každém paketu ověřuje, zda počet polí s názvem *vlan* odpovídá hodnotě 2. Pomocí této podmínky lze odhalit útok *Double VLAN tagging*.

```
condition-type: field-count
field-name: vlan
count: 2
```

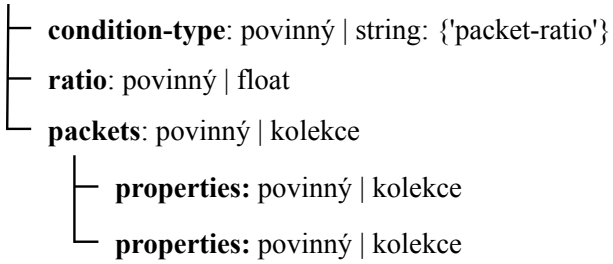
Příklad 8: Zápis podmínky typu field-count odhalující útok Double VLAN tagging.

Podmínka typu packet-ratio

Další podmínka je specifikována pomocí atributu *condition-type* nastaveným na hodnotu *packet-ratio*. Její podstatou je určit poměr počtu dvou různých typů paketů, které uživatel specifikuje. Zápis podmínky kromě jejího typu obsahuje kolekci s názvem **packets**. Tato kolekce slouží pro specifikaci dvou typů paketů, jejichž poměr je třeba zjistit (může tedy obsahovat pouze dvě položky). Položky této kolekce jsou struktury obsahující jeden atribut, což je kolekce s názvem **properties**. Ta slouží k uvedení vlastností jednotlivých typů paketů a její zápis, včetně zpracování, je ekvivalentní jako u stejnojmenného atributu, jehož popis je možné najít v kapitole 3.1. Kromě kolekce *packets* obsahuje zápis podmínky atribut **ratio**, který jako hodnotu obsahuje desetinné číslo vyjadřující poměr počtu prvního typu paketů, vůči počtu druhého typu paketů. Výše uvedené atributy, povinnost jejich zadání, datové typy a jejich vzájemná hierarchie jsou shrnuty na obrázku 3.9.

Zpracování je zahájeno filtrováním obou typů paketů pomocí pravidel uvedených v kolekcích *properties*. U každého typu je určen počet vyfiltrovaných paketů a následně dojde k jejich vzájemnému podělení, přičemž v čitateli zlomku při dělení je uveden počet takového typu paketů, který byl uveden v kolekci *packets* jako první v pořadí. Výsledkem dělení je desetinné číslo, které je porovnáno s hodnotou atributu *ratio*. Pokud je výsledek dělení větší nebo roven hodnotě atributu *ratio*, je podmínka splněna.

Příklad č. 9 uvádí zápis podmínky, pomocí které lze určit poměr počtu paketů, jejichž vlastnosti jsou uvedeny v sekcích *properties*. V tomto případě se jedná o poměr rámců *ARP reply* a *ARP request*, který je možné využít pro detekci útoku *ARP Spoofing*. Poměr



Obrázek 3.9: Shrnutí atributů specifických podmínku typu *packet-ratio*.

je následně porovnán s hodnotou 2.0, která specifikuje dvojnásobné množství rámců *ARP reply* oproti *ARP request*. Při jejím překročení je podmínka vyhodnocena jako pravdivá.

Pro tuto podmínku není zapotřebí definovat ani jeden z prahů *threshold-error* nebo *threshold-warning*. Je to z důvodu, že tato podmínka se netýká konkrétně jednoho paketu, ale k jejímu vyhodnocení je zapotřebí více různých paketů, čili nelze u tohoto typu podmínky hovořit o počtu pozitivních detekcí.

```
condition-type: packet-ratio
ratio: 2.0
packets:
  - properties:
      - field-name: arp.opcode
        attribute: show
        value: 2
  - properties:
      - field-name: arp.opcode
        attribute: show
        value: 1
```

Příklad 9: Podmínka zjišťující poměr paketů ARP reply a ARP request.

Podmínka typu expression

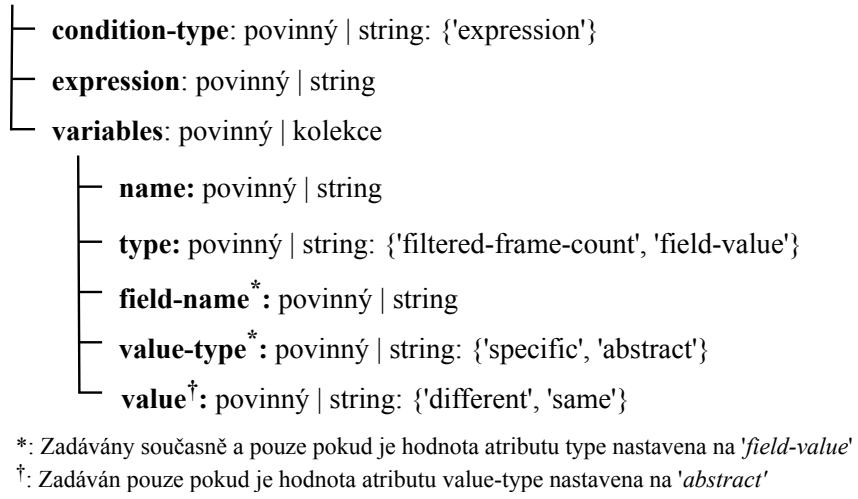
Poslední typ podmínky s názvem *expression* umožňuje vyhodnotit uživatelem zadaný booleovský výraz, jehož syntaxe odpovídá zápisu výrazu v jazyce Python 2.7[8]. Samotný výraz je zadán v řetězcové podobě pomocí atributu **expression**. Jednotlivé proměnné, které jsou ve výrazu užity a jejichž název bude při vyhodnocení nahrazen konkrétní hodnotou, jsou uvedeny v kolekci s názvem **variables**.

Položkami této kolekce jsou struktury, které obsahují atribut **name** označující jméno proměnné, které **musí** vždy začínat velkým písmenem. Následuje typ proměnné, specifikovaný pomocí položky **type**, který může dosahovat hodnot *field-value*, nebo *filtered-frame-count*. V prvním případě se jedná o proměnnou, jejíž název bude při zpracování výrazu nahrazen hodnotou specifikovaného pole uvnitř těla paketu. Následující atributy a jejich podporované hodnoty jsou pak identické jako u podmínky typu *field-value*, včetně podpory abstraktních hodnot. Jediný rozdíl spočívá v tom, že pokud je hodnota atributu *value-type*

[8]Specifikaci výrazů pro jazyk Python 2.7 je možné nalézt na https://docs.python.org/2/reference/expressions.html

nastavena na *specific*, neočekává se atribut *value* a pokud na *abstract*, neočekává se atribut *count*, protože zjištěné hodnoty nejsou porovnávány s těmito atributy, jako u podmínky *field-value*, ale jsou dosazeny jako hodnota proměnné do výrazu.

Pokud je hodnotou atributu type *filtered-frame-count*, neočekávají se žádné další atributy. Odpovídající proměnná ve výrazu je nahrazena hodnotou vyjadřující celkový počet paketů, které byly podmínce předány pro vyhodnocení. Všechny výše uvedené atributy, jejich hierarchie, povinnost zadání a datové typy jsou uvedeny na obrázku 3.10.



Obrázek 3.10: Shrnutí atributů specifických podmínku typu *expression*.

Zpracování podmínky je pak zahájeno získáním všech konkrétních hodnot proměnných, které jsou uvedené v kolekci *variables*. Ty jsou dosazeny do výrazu, který je následně vyhodnocen. Celkový počet paketů, ve kterých výraz nabýval hodnoty *true*, je pak porovnán s hodnotami specifikovaných prahů a případně je vypsáno patřičné hlášení o útoku.

Prahy není zapotřebí definovat v případě, pokud všechny proměnné ve výrazu jsou buď typu *filtered-frame-count*, nebo typu *field-value* s atributem *value-type* nastaveným na *abstract*. V tomto případě se podmínka podobně jako v případě *packet-ratio* netýká pouze jednoho konkrétního paketu, nýbrž paketů více a nelze tedy hovořit o konkrétním počtu pozitivních detekcí. V tomto případě tedy stačí, aby výraz dosáhl hodnoty *true* a podmínka je prohlášena za splněnou.

Příklad č. 10 demonstruje podmínku, která u každého paketu ověří, zda zdrojová *IP* adresa uvedená v hlavičce odpovídá cílové *IP* adrese. Pomocí kolekce *variables* jsou specifikována pole, jejichž hodnoty se mají dosadit za proměnné uvedené ve výrazu v sekci *expression*. Tuto podmínku lze využít k detekci útoku *LAND*.

```
condition-type: expression
expression: SrcIp == DstIp
variables:
  - name: SrcIp
    type: field-value
    field-name: ip.src
    value-type: specific
  - name: DstIp
    type: field-value
    field-name: ip.dst
    value-type: specific
```

Příklad 10: Podmínka typu *expression* odhalující síťový útok *LAND*.

3.2 Požadavky na vstupní data

Jedním z požadavků při tvorbě nástroje bylo rozdělit útoky do dvou skupin podle toho, zda je možné tyto útoky detekovat pouze z jednoho síťového toku v zachycené síťové komunikaci, či nikoliv. Síťovým tokem je míněna obousměrná komunikace od útočníka směrem k oběti (nebo naopak), probíhající mezi konkrétním zdrojovým a cílovým portem prostřednictvím protokolu *TCP* či *UDP*. Většinu útoků lze detekovat pouze z jednoho síťového toku, u některých je to ovšem problém. Jedná se například o útok *Duplicate address detection*, protože komunikaci mezi útočníkem a obětí není možné sdružit do stejného síťového toku.

V tabulce 3.1 je pro každý síťový útok uvedeno, zda k jeho korektní detekci postačuje pouze jeden síťový tok, či je zapotřebí soubor obsahující více různých síťových toků. U útoků probíhajících na linkové vrstvě modelu *TCP/IP* lze implicitně předpokládat, že pro detekci útoku postačuje právě jeden síťový tok, jelikož všechny zaslané rámce jsou, vzhledem k absenci informací ze síťové a transportní vrstvy, klasifikovány do stejného toku nehledě na množství komunikujících entit.

Podpora různých formátů vstupního souboru se zachycenou síťovou komunikací je v tomto případě zcela závislá na verzi používaného nástroje *tshark*. Současná verze (2.3.4) podporuje stejné vstupní soubory jako grafická obdoba tohoto nástroje *Wireshark*[9]. *Tshark* je schopen sám automatizovaně rozpoznat formát vstupního souboru, není tedy třeba explicitně uvádět typ souboru pomocí přípony.

[9]Přehled podporovaných vstupních souborů viz https://wiki.wireshark.org/FileFormatReference

Název útoku	Síťový tok	Kompletní komunikace
Ping of death	✓	
SYN flood		✓
Teardrop	✓	
Land	✓	
DHCP Spoofing		✓
ARP Spoofing	✓	
MAC flooding	✓	
ICMP Redirect	✓	
DAD attack		✓
RA flood		✓
VLAN hopping	✓	
Skenování sítě IPv4	✓	
Skenování sítě IPv6	✓	
Skenování portů		✓

Tabulka 3.1: Shrnutí požadavků na vstupní zachycenou síťovou komunikaci pro korektní detekci útoku.

Kapitola 4

Implementace

Předchozí kapitola se zabývala návrhem obecného formátu pro deklarativní zápis síťových útoků, jehož prostřednictvím je možné popsat způsob detekce těchto útoků. Na základě tohoto návrhu je nyní možné přistoupit k implementaci nástroje, který bude deklarativní zápisy v navrženém formátu interpretovat. Tato kapitola je věnována popisu samotné implementace nástroje, jeho použití, využitých technologií a dalších implementačních detailů.

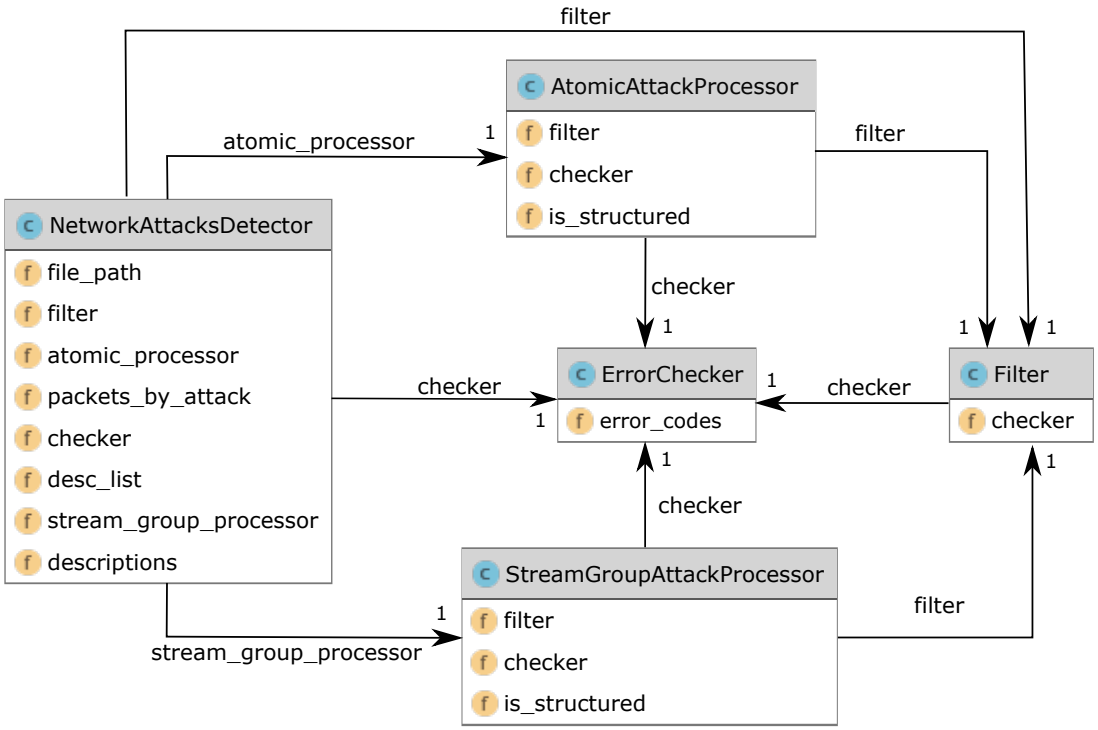
Nástroj je koncipován jako konzolová aplikace implementovaná v objektově orientovaném interpretovaném jazyce Python 2.7. Pro svůj běh nástroj vyžaduje dva vstupy. Prvním z nich je cesta k souboru se zachycenou síťovou komunikací, jenž je převáděn do formátu PDML a ve kterém jsou jednotlivé útoky vyhledávány. Druhým vstupem je pak cesta k adresáři, v němž jsou umístěny soubory ve formátu YAML s deklarativními zápisy síťových útoků, jejichž přítomnost má být zkontrolována ve vstupním souboru se zachycenou komunikací. V adresáři se očekávají pouze soubory s deklarativními popisy, protože nástroj automaticky zpracovává všechny soubory v adresáři uvedené a neprohledává rekurzivně další podadresáře.

Z externích knihoven a nástrojů, které nejsou v základu poskytovány jazykem Python, je pro běh implementovaného nástroje zapotřebí nástroj *tshark*, knihovny *PyYAML*[1] a *cElementTree*. *Tshark* je používán pro převod vstupního souboru do formátu PDML. Zároveň poskytuje prostředky, kterými lze provádět množství optimalizací snižující velikost výstupního PDML souboru, čímž urychluje detekci síťových útoků a snižuje paměťovou náročnost aplikace. Bližšímu popisu těchto optimalizací se věnuje kapitola č. 4.7. Knihovna *PyYAML* je používána k automatizovanému zpracování deklarativních zápisů síťových útoků a *cElementTree* je používána pro parsování a analýzu paketů z PDML souboru.

Nástroj je implementován pomocí objektově orientovaného paradigmatu a je dekomponován celkem do pěti tříd, jejichž názvy jsou následující: *NetworkAttacksDetector*, *AtomicAttackProcessor*, *StreamGroupAttackProcessor*, *Filter* a *ErrorChecker*. *NetworkAttacksDetector* je hlavní třída řídicí zpracování uživatelského vstupu, kontrolu jeho validity a následně delegaci samotné detekce síťových útoků příslušným třídám. Instance tříd *AtomicAttackProcessor* a *StreamGroupAttackProcessor* jsou zodpovědné za korektní detekci všech tří typů síťových útoků *atomic*, *stream* a *group*. Pro filtrování vstupních paketů na základě uživatelem specifikovaných pravidel v deklarativních zápisech síťových útoků je využita třída *Filter*. Poslední třída *ErrorChecker* je zodpovědná za ověřování validity uživatelského vstupu a výpis chybových hlášení. Vztahy jednotlivých tříd jsou vizualizovány ve třídním diagramu na obrázku 4.1. Následující podkapitoly 4.1 až 4.5 jsou věnovány detailnějšímu po-

[1] Viz. http://pyyaml.org/wiki/PyYAML

pisu jednotlivých tříd a samotnému popisu procesu detekce síťových útoků. Podkapitola 4.6 se zabývá popisem formátu, který je použit pro výpis informací o pozitivní či negativní detekci jednotlivých útoků. Poslední podkapitola 4.7 se zabývá popisem optimalizací, které byly implementovány za účelem snížení paměťové náročnosti nástroje.



Obrázek 4.1: Třídní diagram vizualizující vztah jednotlivých implementovaných tříd.

4.1 Třída NetworkAttacksDetector

Implementaci třídy *NetworkAttacksDetector* je možné nalézt v souboru s názvem *na_detector.py*. Ten zároveň obsahuje definici metody `main`. Jedná se tedy o skript, který je využíván pro spuštění samotného nástroje. Na počátku skriptu jsou zpracovány argumenty příkazové řádky pomocí knihovny *argparse*, které jsou následně předány jako parametr konstruktoru při vytváření objektu třídy *NetworkAttacksDetector*. Vytvořená instance následně ověří validitu zadaných parametrů, k čemuž využívá objekt třídy *ErrorChecker*. Při tom je kontrolována existence vstupního souboru a adresáře s deklarativními zápisy. Obsah adresáře je poté procházen a jednotlivé zápisy jsou parsovány pomocí knihovny *PyYAML*. Po procesu parsování jsou u každého deklarativního zápisu zkontrolovány jeho syntaktické náležitosti s využitím již zmíněné instance třídy *ErrorChecker*. Pokud zápis nedodržuje požadavky navrženého deklarativního formátu, je daný zápis vyloučen ze zpracování.

Před samotnou detekcí síťových útoků je zapotřebí provést konverzi vstupního souboru se zachycenou komunikací do formátu PDML pomocí nástroje *tshark*. Příkaz, kterým je konverze realizována, je uveden v příkladu 11. Po dokončení konverze je již možné načítat a zpracovávat jednotlivé pakety pomocí knihovny *cElementTree*. Pakety v konvertovaném souboru jsou tedy sekvenčně procházeny a postupně parsovány. Na základě deklarativních

zápisů je pak pro každý útok vytvořen seznam paketů, které jsou nezbytné pro detekci konkrétního útoku. K tomuto účelu je využito instance třídy *Filter*, která ověřuje, zda pro daný paket platí uživatelem specifikované podmínky v aktuálně zpracovávaném deklarativním popisu. Poté, co jsou všechny pakety rozřazeny ke konkrétním útokům, je zpracování jednotlivých popisů delegováno na instance tříd *AtomicAttackProcessor*, či *StreamGroupAttackProcessor* na základě typu daného síťového útoku. Tyto instance pak útok na odpovídající množině paketů vyhodnotí a zahlásí jeho pozitivní detekci či hlášení o jeho nenalezení.

```
tshark -r nazev_vstupniho_souboru -T pdml
```

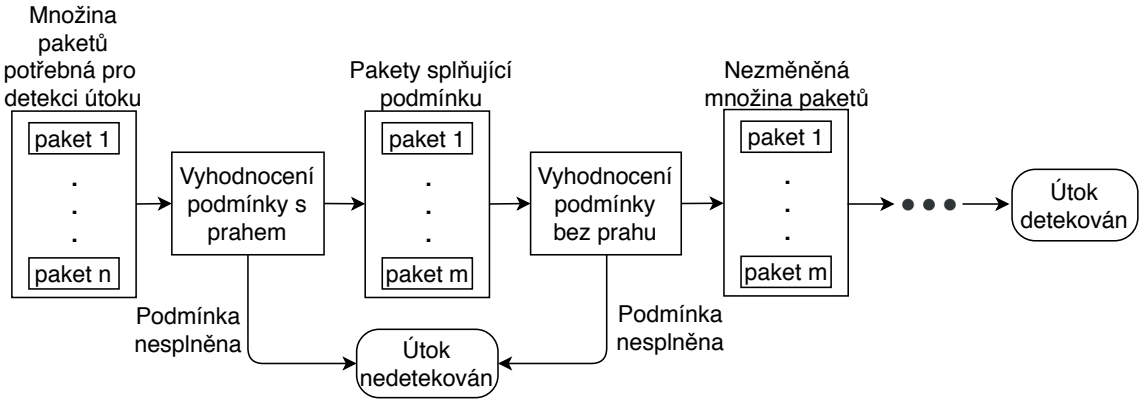
Příklad 11: Spuštění nástroje *tshark* realizující konverzi do formátu PDML.

4.2 Třída AtomicAttackProcessor

Implementaci třídy *AtomicAttackProcessor* je možné nalézt v souboru *atomic_attack_processor.py*. Instance této třídy má odpovědnost za vyhodnocení deklarativních popisů pro útoky typu *atomic*, jejichž zpracování je započato voláním metody `process`, které je jako parametr předán samotný popis útoku a vytvořený seznam paketů, které jsou pro daný útok relevantní. Následně je procházena kolekce s detekčními podmínkami, které jsou postupně vyhodnocovány. Pokud je uvedeno více detekčních podmínek, je způsob jejich vyhodnocení přizpůsoben tomu, zda musí platit všechny uvedené podmínky, či postačuje platnost pouze jedné z nich.

Pokud musí platit všechny podmínky současně, je nejdříve vyhodnocena první podmínka v seznamu a ze vstupních paketů se vyfiltrují pouze takové pakety, pro které tato podmínka platí. Tyto pakety jsou pak předány jako vstup při vyhodnocení následující podmínky a takto se pakety postupně filtrují až dokud není vyhodnocena poslední podmínka. Jestliže je v seznamu detekčních podmínek uvedena podmínka, která nepotřebuje definovat hodnoty prahů (výstupem jejího vyhodnocení je pouze podmínka splněna, nebo nesplněna, nikoliv seznam paketů označených za pozitivní detekci), je tato podmínka vyhodnocena nad předanou množinou paketů. Pokud je splněna, je tato množina bez modifikace předána jako vstup pro vyhodnocení další podmínky. Po vyhodnocení všech podmínek je celkový počet pozitivních detekcí, pro které platí všechny uvedené podmínky, porovnán s hodnotami uvedených prahů. Na základě porovnání jsou vypsána příslušná hlášení tak, jak je uvedeno v kapitole č. 3.1. Výše uvedený proces je vizualizován na obrázku 4.2. Součástí kladných hlášení o detekci síťového útoku jsou pak i čísla paketů, které byly označeny za pozitivní detekci. Pokud **všechny** podmínky uvedené v seznamu detekčních podmínek nevyžadují zadání prahu, je při jejich splnění vypsáno varovné hlášení o potenciálním nalezení útoku (díky absenci prahů nelze explicitně určit, zda vypsat varovné, či chybové hlášení o útoku). Součástí hlášení pak není seznam paketů, ze kterých byl útok detekován, protože jejich uvedení nemá v daném kontextu žádnou vypovídající hodnotu.

Jestliže postačuje platnost pouze jedné podmínky uvedené v seznamu detekčních podmínek, je pak po vyhodnocení každé z nich počet pozitivních detekcí porovnán s prahy, které jsou přímo součástí zápisu každé podmínky (příp. dojde pouze k vyhodnocení podmínky, pokud nevyžaduje zadání prahů). Je-li podmínka splněna, další podmínky se již neověřují a útok je prohlášen za detekovaný. Na rozdíl od vyhodnocení, kdy musí platit všechny podmínky v seznamu, je zde každá podmínka vyhodnocena nad původní množinou relevantních paketů a nedochází zde k postupnému filtrování jako v předchozím případě.



Obrázek 4.2: Proces vyhodnocování detekčních podmínek, pokud musí platit všechny detekční podmínky. Pro hodnoty n a m platí $n \geq m$.

4.3 Třída StreamGroupAttackProcessor

Implementace této třídy je umístěna v souboru *stream_group_attack_processor.py*. Útoky typu *stream* a *group* se od sebe navzájem liší pouze způsobem, jakým jsou pakety rozřazovány do skupin, ve kterých se hledá přítomnost síťového útoku. Z tohoto důvodu je zpracování obou typů útoků umístěno do společné třídy. Zpracování útoku je zahájeno rozčleněním paketů do jednotlivých skupin. V případě útoku typu *stream* jsou pakety seskupeny do *TCP* a *UDP* streamů na základě indexu, které do paketů doplňuje automaticky nástroj *tshark*. Pokud pakety tento index neobsahují, jsou při zpracování útoku ignorovány. U útoku typu *group* jsou postupně procházena pole uvedená v deklarativním popisu a pokud je aktuálně vyhledávané pole nalezeno, je paket umístěn do skupiny na základě jeho hodnoty. Pakety, ve kterých se nenajde žádné ze specifikovaných polí, jsou při detekci ignorovány.

V první fázi detekce útoku je ověřeno, zda se v aktuálně prohledávané skupině vyskytují všechny typy paketů, které jsou uvedeny v deklarativním popisu. Poté dojde k ověření, zda odpovídá jejich vzájemný poměr specifikovaný pro každý typ paketů pomocí atributů *count* (či dvojicí *min-count* a *max-count*). Ze skupiny jsou tedy vyfiltrovány jednotlivé typy paketů uvedené v deklarativním zápisu, čímž jsou vytvořeny podskupiny paketů, kde jedna podskupina obsahuje právě jeden typ paketů. Následně je ověřeno, zda poměr paketů v jednotlivých podskupinách odpovídá informacím uvedeným v zápisu útoku.

Při ověřování vzájemného poměru paketů je nezbytné uvažovat i případ, kdy se v aktuálně prohledávané skupině vyskytuje stejný síťový útok vícenásobně. Kvůli tomu je nezbytné upravit počty jednotlivých typů paketů uvedených v deklarativním popisu. To je řešeno tak, že se v deklarativním zápisu útoku najde první typ paketů, který má přesně specifikovaný počet, kolikrát se má ve skupině vyskytovat (pomocí atributu *count*). Následně se zkontroluje skutečný počet paketů daného typu v odpovídající vyfiltrované podskupině, a pokud se tento počet liší od hodnoty uvedené v deklarativním popisu, je předpokládáno, že se útok vyskytuje ve skupině vícekrát. Počet paketů (včetně počtu ostatních typů paketů) uvedený v deklarativním zápisu se tedy patřičně upraví vynásobením koeficientem. Ten je vypočítaný jako podíl skutečného počtu paketů v dané podskupině a počtu uvedeného v deklarativním zápisu pro daný typ paketů. Následně se ověří počet paketů v jednotlivých podskupinách, který pro pozitivní detekci musí odpovídat těmto upraveným počtům. Pokud žádný typ paketů nemá přesně specifikovaný počet výskytů v deklarativním zápisu (tzn. všechny typy

paketů mají počet specifikovaný dvojicí atributů *min-count* a *max-count*), nejsou počty jednotlivých typů paketů nijak upravovány, protože není možné přesně stanovit koeficient, kterým by se měly jednotlivé počty vynásobit.

Pokud jsou atributy *follow* a *specified-only* nastaveny na hodnotu *false*, je ověřením výskytu jednotlivých typů paketů a jejich poměru detekce ukončena, a pokud výskyt i poměr odpovídá, je skupina označena za pozitivní detekci. Jestliže je vyžadováno, aby jednotlivé typy paketů, uvedené v deklarativním zápisu, následovaly ve skupině bezprostředně za sebou (atribut *follow* je nastavený na hodnotu *true*), je následně ověřeno, zda tyto pakety na sebe ve skupině skutečně bezprostředně navazují. Pokud je u paketů nějakého typu uvedeného v zápisu očekáván více než jeden paket, musí na sebe navazovat i odpovídající počet těchto paketů. Pokud je atribut *specified-only* rovněž nastaven na hodnotu *true*, je navíc zkontrolováno, zda se mimo specifikované typy paketů nevyskytují ve skupině i jiné pakety. Pokud platí všechny výše uvedené podmínky, je daná skupina prohlášena za pozitivní detekci. Po prohledání všech skupin je počet pozitivních detekcí porovnán s uvedenými prahy a na základě výsledku porovnání je vypsáno příslušné hlášení. Pokud byl útok detekován, jsou vypsány i skupiny paketů, které byly označeny za pozitivní detekci.

4.4 Třída Filter

Třída *Filter* je implementována v souboru *filter.py*. Odpovědnost této třídy spočívá ve filtrování paketů ze vstupního souboru na základě pravidel uvedených v deklarativních popisech síťových útoků. Třída poskytuje operaci, jejímž vstupem je kolekce paketů a seznam filtrujících pravidel zapsaných dle syntaktických požadavků uvedených v kapitole 3.1. Výstupem je pak kolekce paketů, pro které platí všechna pravidla předaná jako vstupní parametr. Dále obsahuje operaci, jejímž vstupem je jeden konkrétní paket a seznam filtračních pravidel a výstupem je informace o tom, zda paket všechna uvedená pravidla splňuje. Tato operace se využívá při rozřazování paketů do skupin paketů potřebných pro detekci jednotlivých síťových útoků.

4.5 Třída ErrorChecker

Implementaci třídy *ErrorChecker* je možné nalézt v souboru *error_checker.py*. Třída implementuje metody používané pro validaci uživatelského vstupu a výpis chybových hlášení s tím spojený. Po spuštění nástroje zkontroluje existenci zadaného adresáře s deklarativními zápisy útoků a vstupního souboru se zachycenou síťovou komunikací. Před samotným procesem detekce síťových útoků kontroluje syntaktickou správnost jednotlivých deklarativních zápisů. Součástí této kontroly je ověření přítomnosti všech požadovaných atributů v zápisu, jejich datových typů a případně kontrola hodnot, kterých mohou dané atributy nabývat. Pokud některý z deklarativních zápisů není vytvořen zcela syntakticky správně, je zahlášena příslušná chyba a zápis je vyloučen z dalšího zpracování.

4.6 Formát výstupu programu

O výstup programu, jehož cílem je informovat o pozitivní, či naopak negativní detekci síťových útoků, se starají třídy, které jednotlivé útoky vyhodnocují – *AtomicAttackProcessor* a *StreamGroupAttackProcessor*. Nástroj poskytuje celkem dva typy výstupů, mezi kterými lze přepínat pomocí parametru zadaného při spuštění programu. První z nich je

prezentován v lidsky čitelnější textové podobě a druhý je strukturovaně zapsaný pomocí značkovacího jazyka XML. Oba výstupy vždy vypíší název útoku, informaci o tom, zda byl detekován a úroveň vyjadřující závažnost detekce (varování, či hlášení o chybě). Pro útok typu *atomic* také seznam paketů, ze kterých byl útok detekován. Tento seznam je ovšem uveden pouze v případě, že detekční podmínky daného útoku vyžadují zadání prahů detekce (viz. kapitola 3.1.5). U útoku typu *stream* jsou součástí výpisu čísla síťových toků, které byly označeny za pozitivní detekci. Spolu s nimi je pro každý tok uvedena informace, zda se jedná o *TCP* nebo *UDP* stream. V případě útoku typu *group* je vypsán seznam skupin, které byly označeny za pozitivní detekci společně s čísly paketů, které danou skupinu tvoří.

Strukturovaný zápis pomocí XML je tvořen kořenovou párovou značkou s názvem *report*, jejímž obsahem jsou značky s označením *attack*, které reprezentují hlášení o pozitivní či negativní detekci právě jednoho útoku. Součástí značky *report* je atribut *file*, jehož hodnotou je textový řetězec reprezentující název vstupního souboru, pro nějž je výstup tvořen. Každá značka s názvem *attack* obsahuje atribut *name*, jehož hodnotou je textový řetězec označující název útoku. Dalším atributem je *type*, který označuje typ daného útoku. Tento atribut může tedy dosahovat pouze řetězcových hodnot *atomic*, *stream* a *group*. Další atribut *detection* může dosahovat pouze dvou hodnot. První z nich je textový řetězec *positive*, který označuje pozitivní detekci útoku a druhý *negative* detekci negativní. Pokud byl útok detekován (atribut *detection* má hodnotu *positive*), je přítomen i další atribut *level*, který označuje závažnost detekce na základě porovnání s prahy uvedenými v deklarativním popisu. Může tedy dosahovat pouze hodnot *error* a *warning*.

Pokud je typ útoku *atomic* a součástí výstupu je i seznam paketů, pomocí nichž byl útok detekován, je XML značka *attack* párová a jejím obsahem jsou značky s názvem *packet*. Tyto značky obsahují pouze jeden atribut s názvem *num*, které označují číslo paketu ve vstupním souboru. Pokud součástí výstupu pro tento typ útoku seznam paketů není, je značka *attack* nepárová. Příklad výstupu programu pro útok typu *atomic* je uveden v příkladu 12.

```
<?xml version="1.0" encoding="UTF-8"?>
<report file="teardrop.pcap">
        <attack name="Teardrop" type="atomic" detection="positive" level="error">
                <packet num="11"/>
        </attack>
</report>
```

Příklad 12: Výstup nástroje pro pozitivní detekci síťového útoku typu *atomic*.

Součástí výstupu pro útok typu *stream* je vždy i seznam síťových toků, které byly označeny na pozitivní detekci. Značka *attack* je tedy vždy párová a obsahuje značky s názvem *stream*, které mají vždy dva atributy. Podobně jako u předchozího typu útoku je prvním z nich atribut *num* označující číslo síťového toku. Druhým je atribut type, jenž dosahuje hodnot pouze *tcp*, nebo *udp* a označuje tak typ síťového toku. Příklad zkráceného výstupu programu pro útok typu *stream* je uveden v příkladu 13.

```
<?xml version="1.0" encoding="UTF-8"?>
<report file="syn_flood.pcap">
        <attack name="SYN flood" type="stream" detection="positive" level="error">
                <stream type="tcp" num="0" />
                <stream type="tcp" num="1" />
        </attack>
</report>
```

Příklad 13: Výstup nástroje pro pozitivní detekci síťového útoku typu *stream*.

Výstup útoku typu *group* vždy obsahuje i seznam skupin, které byly označeny za pozitivní detekci, včetně specifikace paketů, které danou skupinu tvoří. Skupina je reprezentována značkou s názvem *group*, která obsahuje pouze jeden atribut *id*, jehož hodnotou je textový řetězec reprezentující identifikátor dané skupiny. Tento identifikátor je hodnota pole v paketu, na základě které byly pakety umístěny do stejné skupiny. Značka *group* je párová a obsahuje seznam paketů, které danou skupinu tvoří. Syntaxe zápisu paketů je stejná jako v útoku typu *atomic*. Příklad výstupu programu pro typ útoku *group* je uveden v příkladu 14.

```
<?xml version="1.0" encoding="UTF-8"?>
<report file="dad.pcap">
        <attack name="DAD" type="group" detection="positive" level="warning">
                <group id="fe80::a00:27ff:fe66:1c4a">
                        <packet num="7"/>
                        <packet num="8"/>
                        <packet num="9"/>
                </group>
        </attack>
</report>
```

Příklad 14: Výstup nástroje pro pozitivní detekci síťového útoku typu *group*.

4.7 Optimalizace

Po dokončení implementace nástroje a jeho testování na vstupních souborech o větších velikostech bylo zjištěno, že nástroj nebyl schopen tyto soubory zpracovat z důvodu kompletního vyčerpání kapacity operační paměti. Unixové operační systémy při zaplnění paměti obvykle využívají pro ukládání přebytečných dat diskové uložiště, což vede k výraznému zpomalení procesu detekce síťových útoků a nástroj se tak stal prakticky nepoužitelným. K vyčerpání operační paměti docházelo kvůli značné velikosti konvertovaných PDML souborů, která v mnohých případech dosahovala hodnot několika gigabytů. Obrovská velikost konvertovaného PDML souboru oproti vstupnímu souboru se zachycenou síťovou komunikací je způsobena tím, že vstupní soubor je tvořen proudem bytů, jejichž význam je následně interpretován paketovým analyzátorem, zatímco formát PDML obsahuje již interpretovaná data v textové podobě. Kromě samotných hodnot polí paketu tedy PDML soubor obsahuje i názvy těchto polí a další informace interpretované paketovým analyzátorem. Hodnoty jednotlivých polí jsou navíc často uvedeny v několika různých formátech současně (např. ve formě raw bytů, textových řetězců, hexadecimálních zápisů bytů, apod.). Určitý nárůst velikosti je taktéž spojen s vlastnostmi jazyka XML, na kterém je formát PDML založen. Mezi ně patří například existence párových značek, u kterých je nezbytné pro každou otevírací

značku uvést její ukončující značku apod. Uvnitř paketů jsou také často přenášena obsáhlá aplikační data, která jsou pro detekci síťových útoků nepodstatná, ale mají velmi výrazný vliv na celkovou velikost zpracovávaného souboru (např. obsah přenášený protokoly HTTP, FTP a podobně).

Při zpracování PDML souborů pomocí knihoven pro parsování XML dokumentů navíc jejich velikost v operační paměti několikanásobně vzroste, kvůli nutnosti uchování informací o návaznosti jednotlivých XML uzlů (pro každý uzel se uchovává informace o jeho předcích, následnících, dětských uzlech, atp.). Ze všech výše uvedených důvodů je tedy prakticky nereálné zpracovávat soubory o vyšší velikosti s využitím současných výpočetních prostředků čistě v operační paměti bez využití diskového uložiště, což vede ke značnému zpomalení procesu detekce síťových útoků. Z tohoto důvodu je nezbytné snížit velikost konvertovaných PDML souborů pomocí několika optimalizací.

První optimalizací je redukce počtu polí v paketu pouze na ta pole, která jsou nezbytná pro úspěšnou detekci útoku. Před samotnou konverzí vstupního souboru do formátu PDML jsou tedy postupně procházeny všechny deklarativní zápisy útoků, ze kterých je vytvořen seznam potřebných polí. Nástroj *tshark* následně umožňuje specifikovat (pomocí přepínače `-e`[2]), která pole mají být zahrnuta v konvertovaném výstupu, díky čemuž dojde k výrazné redukci velikosti konvertovaného souboru. Tato operace má rovněž pozitivní vliv na rychlost konverze, protože velké množství polí paketu nemusí být vůbec interpretováno. Při použití přepínače `-e` je ve výsledném PDML souboru u každého pole v paketu uvedena pouze hodnota atributu *show*[3] a tento atribut je přejmenován na *value*.

Dále je možné výstupní velikost konvertovaného souboru optimalizovat pomocí filtru, jenž umožňuje specifikovat (pomocí přepínače `-Y`[2]), které pakety mají být zahrnuty ve výstupu konverze. Tento filtr je předán jako parametr nástroji *tshark* před samotnou konverzí a jeho syntaxe je naprosto identická jako u display filterů[4], které jsou využívány v nástroji *Wireshark*. Filtr je vytvořen na základě informací uvedených v kolekci *properties* v deklarativních zápisech útoků. Z jednotlivých filtrovacích pravidel v této kolekci jsou tedy získány názvy polí a případně jejich hodnoty, ze kterých je sestaven textový řetězec reprezentující display filter. Díky tomuto kroku je zajištěno, že ve výstupním konvertovaném souboru nebudou přítomny pakety, které jsou pro detekci síťových útoků nepotřebné.

Obě výše uvedené optimalizace jsou implementovány ve třídě *NetworkAttacksDetector*. Tyto optimalizace redukují množství polí v paketech a samotný počet těchto paketů na nejnižší možnou úroveň potřebnou pro detekci síťových útoků. Příkaz, kterým je realizována konverze do formátu PDML zahrnující obě výše uvedené optimalizace je uveden v příkladu 15.

```
tshark -r nazev_souboru -T pdml -e ip.src -e ip.dst ... -Y display_filter
```

Příklad 15: Spuštění nástroje *tshark* realizující optimalizovanou konverzi do formátu PDML.

Další optimalizací, která byla implementována, je iterativní parsování paketů namísto zpracování celého konvertovaného souboru najednou. Soubor je manuálně procházen řádek po řádku a jednotlivé řádky jsou postupně konkatenovány, dokud není načten úsek konvertovaného souboru reprezentující právě jeden paket, který je následně pomocí knihovny

[2]Popis viz. https://www.wireshark.org/docs/man-pages/tshark.html

[3]Bez použití přepínače `-e` je hodnota uvedena v několika různých formátech pomocí různých XML atributů.

[4]Viz. https://wiki.wireshark.org/DisplayFilters

cElementTree parsován. Díky této optimalizaci je šetřena operační paměť kvůli tomu, že nemusí být ukládány informace o vzájemné návaznosti paketů (informace o předcích, následnících, apod.) ve vstupním souboru, které jsou pro proces detekce nepotřebné. Tento postup může být aplikován především kvůli tomu, že lze očekávat validní PDML soubor s identickou strukturou při opakovaných konverzích pomocí nástroje *tshark*.

Implementované optimalizace zároveň zajišťují i určitou variabilitu v použití nástroje pokud zařízení, na kterém je nástroj spouštěn, disponuje nižší kapacitou operační paměti. Pokud je zapotřebí snížit paměťovou náročnost nástroje, je možné ho spouštět vícekrát po menších počtech deklarativních zápisů. Díky výše uvedeným optimalizacím se tak redukuje celkový počet polí zpracovávaných během jednoho běhu nástroje stejně tak jako celkový počet paketů, které jsou pro detekci síťových útoků zapotřebí. To vede ke snížení paměťových nároků nástroje, což je často důležité při zpracování vstupních souborů s větším počtem zachycených paketů.

Díky implementaci těchto optimalizací je tedy nástroj schopen zpracovávat vstupní soubory o výrazně větších velikostech, než bez jejich použití. V níže uvedené tabulce 4.1 je uvedeno porovnání velikostí výstupů konverze do formátu PDML před a po aplikování výše uvedených optimalizací. První dva sloupce určují velikost vstupního souboru a počet paketů v něm obsažených. Třetí a čtvrtý sloupec udává velikost PDML souboru bez optimalizací a s optimalizacemi. Poslední sloupec pak udává procentuální rozdíl ve velikostech. Z tabulky je možné určit, že velikost výstupního souboru byla redukována přibližně o 95 %.

Velikost vstupu [MB]	**Počet paketů [tis.]**	**Velikost PDML [MB]**	**Velikost opt. PDML [MB]**	**Zmenšení velikosti o [%]**
3,4	29	320	28	91,3
56	94	2200	104	95,3
98	142	3500	156	95,5
153	193	4100	192	95,3
202	228	5000	256	94,9
275	282	6400	315	95,1
352	402	8300	450	94,6
453	443	9800	498	94,9

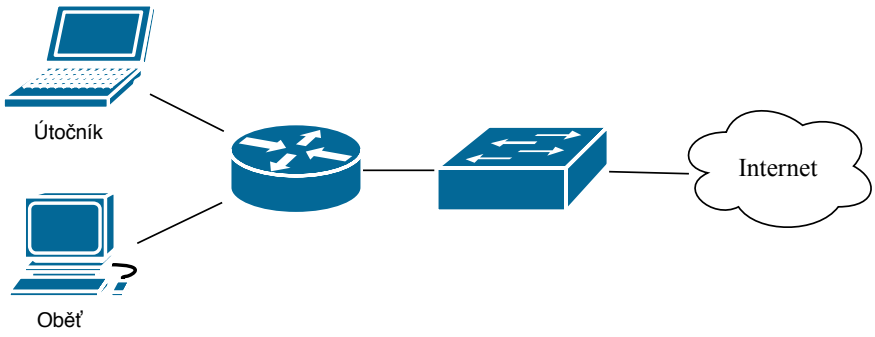
Tabulka 4.1: Porovnání velikostí výstupů konverze do formátu PDML před a po aplikování optimalizací.

Kapitola 5

Testování a zhodnocení

Po implementaci nástroje a jeho následné optimalizaci, která výrazně snížila jeho paměťovou náročnost, bylo možné přistoupit k samotnému testování nástroje. Testování probíhalo celkem ve třech fázích, přičemž první dvě fáze byly věnovány testování nástroje z hlediska jeho funkčnosti a třetí fáze testům jeho výkonnosti. V první fázi testování bylo nezbytné ověřit, zda všechny síťové útoky popsané v deklarativních zápisech jsou korektně detekovány při analýze vstupních souborů, které dané síťové útoky obsahují. K tomu bylo zapotřebí získat soubory se zachycenou síťovou komunikací, v nichž jsou útoky přítomny. Některé takové soubory bylo možné získat z různých veřejných databází na internetu, nicméně většinu útoků bylo zapotřebí reálně provést a zachytit pomocí některého z paketových analyzátorů.

Pro tento účel bylo nezbytné vytvořit testovací prostředí, ve kterém je možné síťové útoky generovat bez ovlivnění reálného chodu sítě a provozu dalších uživatelů. Zachytávání síťových útoků tedy probíhalo ve virtuálním testovacím prostředí vytvořeným pomocí nástroje *VirtualBox* a reálné domácí síti sestávající ze stanice oběti, útočníka a routeru. Topologie domácí sítě je vizualizována na obrázku 5.1. Pro samotné generování útoků bylo využito množství volně dostupných nástrojů (např. *THC-IPV6*, *nmap*, *Metasploit*, *ettercap*, apod.), či paketový manipulátor *Scapy*[1] pro jazyk Python.



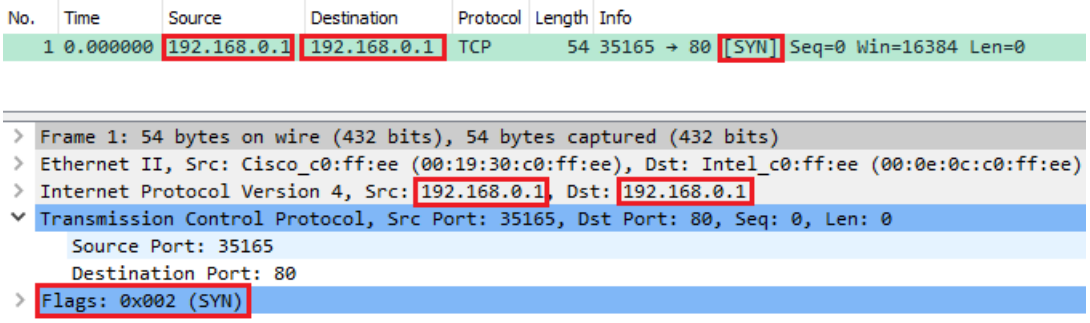
Obrázek 5.1: Topologie sítě při realizaci síťových útoků za účelem zachycení síťové komunikace.

[1] Viz. http://www.secdev.org/projects/scapy/

5.1 Testování korektní detekce síťových útoků

Samotné testování v první fázi probíhalo tak, že pro každý deklarativní popis útoku byl spuštěn implementovaný nástroj, jehož vstupem byl soubor se zachycenou komunikací obsahující síťový útok, který daný deklarativní zápis popisoval. Přítomnost útoku ve vstupním souboru byla ověřena manuální kontrolou souboru pomocí paketového analyzátoru *Wireshark*. Po dokončení analýzy vstupního souboru implementovaným nástrojem s využitím konkrétního deklarativního zápisu byl pak zkontrolován výstup nástroje, od kterého bylo očekáváno zahlášení pozitivní detekce síťového útoku.

Konkrétní příklad testování nástroje bude demonstrován na síťovém útoku *Local Area Network Denial* (*LAND*), jehož princip spočívá v zaslání paketu s nastaveným příznakem *SYN*, ve kterém je zdrojová i cílová IP adresa shodně nastavena na IP adresu oběti. Po přijetí takového paketu se oběť snaží ustavit spojení sama se sebou, což vede k zamrznutí nebo pádu systému (bližší informace viz. kapitola 2.4). Na obrázku 5.2 je zobrazen redukovaný výstup nástroje *Wireshark* ukazující obsah souboru, ve kterém je paket způsobující takový útok přítomen. Pro jednodušší demonstraci je ve výstupu zobrazen pouze paket, který útok *LAND* způsobuje.



Obrázek 5.2: Výstup nástroje *Wireshark* pro soubor obsahující síťový útok *LAND*. V červeném rámečku jsou zvýrazněna důležitá pole v paketu.

Na tomto vstupním souboru byl pak implementovaný nástroj spuštěn spolu s deklarativním popisem útoku *LAND*. Výstup programu ohlašující pozitivní detekci útoku je uveden v příkladu 16. Součástí výstupu je i závažnost detekce (atribut *level*) a číslo paketu, ze kterého byl útok detekován. Stejným způsobem bylo postupně otestováno všech 23 deklarativních popisů, čímž bylo ověřeno, že nástroj je schopen síťové útoky pomocí deklarativních popisů detekovat.

```
<?xml version="1.0" encoding="UTF-8"?>
<report>
        <attack name="LAND" type="atomic" detection="positive" level="error">
                <packet num="1"/>
        </attack>
</report>
```

Příklad 16: Výstup nástroje potvrzující pozitivní detekci síťového útoku *LAND*.

5.2 Testování false positives

Druhá fáze testování byla zaměřena na testování tzv. *false positives* [27], což jsou situace, ve kterých je zahlášena pozitivní detekce útoku i při běžné síťové aktivitě. Tyto jevy jsou nežádoucí, protože zvyšují vytížení síťových administrátorů, kteří musí síťový provoz analyzovat, aby rozlišili, zda se jedná o skutečnou hrozbu, nebo o falešnou detekci. Testování nástroje zaměřené na *false positives* je vhodné provádět na vstupních souborech obsahujících reálný síťový provoz zachycený v různých síťových prostředích, které jsou svojí strukturou a velikostí podobné reálným sítím, ve kterých by mohl být implementovaný nástroj potenciálně využíván. Pro tento účel bylo z veřejných internetových databázích staženo množství souborů se zachyceným reálným síťovým provozem, ve kterých se rozsah paketů pohyboval v rozsahu od 5 000 do 500 000. Zároveň bylo v testovacím síťovém prostřední zachyceno několik souborů s různým síťovým provozem, aby testování pokrývalo co největší záběr odlišných typů provozu.

Samotné testování implementovaného nástroje na *false positives* probíhalo tak, že pro každý vstupní soubor byl spuštěn nástroj se všemi vytvořenými deklarativními popisy. Po dokončení forenzní analýzy všech těchto souborů byly zkontrolovány jednotlivé výstupy nástroje. Pokud byl některý ze zkoumaných útoků detekován, došlo k ověření, zda se útok v souboru skutečně vyskytuje, nebo zda se jedná o falešnou detekci. Jestliže došlo k zahlášení falešné detekce, byla zjištěna její příčina a na základě realizovaných testů pak byly provedeny úpravy deklarativních zápisů síťových útoků, či rozšíření formátu pro jejich zápis tak, aby bylo eliminováno co nejvíce těchto falešných detekcí. Po provedení prvních dvou fází bylo tedy ověřeno, zda všech 23 deklarativních zápisů je vhodně navrženo a je pomocí nich možné korektně detekovat jimi popisované síťové útoky.

5.3 Výkonnostní testování nástroje

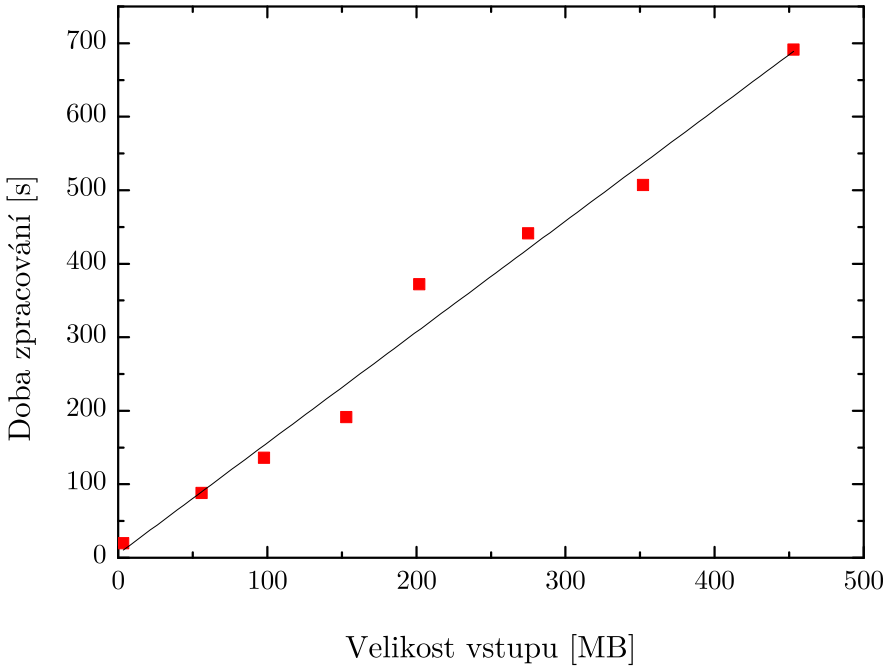
Třetí fáze byla věnována testování výkonnosti implementovaného nástroje na stejných souborech se zachycenou komunikací, které byly získány pro testování *false positives*. V tabulce 5.1 je uvedeno porovnání doby běhu a paměťových nároků nástroje pro různou velikost vstupních souborů obsahujících různý počet paketů. V rámci jednoho běhu nástroje musí dojít ke konverzi vstupního souboru do formátu PDML. Následně musí být pomocí knihovny *cElementTree* iterativně zpracovány jednotlivé pakety z výstupu konverze. Pro každý popis útoku je pak vytvořen seznam relevantních paketů potřebných pro jeho detekci. V poslední části je pak vyhodnoceno 23 deklarativních popisů útoků. Hodnoty v tabulce 5.1 byly naměřeny na zařízení *Lenovo ThinkPad EDGE E531* s procesorem Intel Core i5-3230M (2.6/3.2 GHz) a operační pamětí 8GB.

Z tabulek 4.1 a 5.1 je patrné, že na současných průměrných počítačích je možné analyzovat vstupní soubory až o velikostech přibližně půl gigabytu, přičemž vyhodnocení 23 různých síťových útoků na takto velkém vstupu zabere přibližně 12 minut. Na základě údajů z tabulky 5.1 je možné stanovit, že implementovaný nástroj dosahuje při svém běhu propustnosti přibližně 5,2 Mb/s. Velikost vstupních souborů, které je nástroj schopen zpracovat je limitován především kapacitou operační paměti, protože při zpracování souborů o velikosti půl gigabytu již dochází k zaplnění přibližně 8 GB paměti RAM. Použití nástroje je tedy nejvíce ovlivněno právě kapacitou RAM. Pokud dojde k vyčerpání dostupné kapacity operační paměti, unixové operační systémy začínají využívat diskového uložiště pro ukládání nadbytečných dat, což vede ke značnému zpomalení při detekci. Pokud je tedy zapotřebí zpracovávat soubory o větších velikostech, je nezbytné rozšířit kapacitu operační paměti

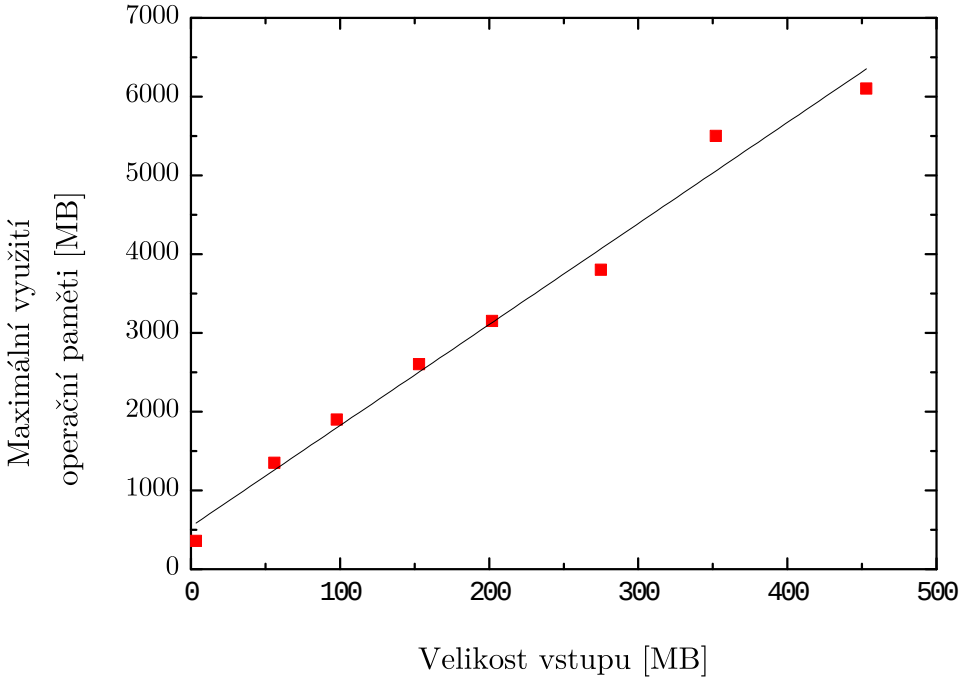
Velikost vstupu [MB]	Počet paketů [tis.]	Doba zpracování [s]	Max. využití oper. paměti [MB]
3,4	29	19,5	358
56	94	88	1350
98	142	136	1900
153	192	191	2600
202	228	372	3150
275	282	441	3800
352	402	507	5500
453	443	691	6100

Tabulka 5.1: Porovnání doby běhu a paměťové náročnosti nástroje pro různé velikosti vstupních souborů se zachycenou síťovou komunikací.

nebo využít disků typu SSD, na kterých zpomalení již není tolik výrazné. Závislost velikosti vstupního souboru na době potřebné pro jeho zpracování a využití operační paměti je vizualizována pomocí grafů na obrázcích 5.3 a 5.4. Z grafů je možné určit, že tato závislost je přibližně lineární, díky čemuž je možné konstatovat, že využití paměti a čas zpracování bude s velikostí vstupního souboru růst taktéž lineárně.



Obrázek 5.3: Závislost velikosti vstupního souboru na době potřebné k jeho zpracování.



Obrázek 5.4: Závislost velikosti vstupního souboru na využití RAM během jeho zpracování.

Kapitola 6

Závěr

Cílem této diplomové práce bylo navrhnout a implementovat nástroj realizující detekci síťových útoků ze zachycené síťové komunikace s využitím nástroje *tshark*. Význam nástroje *tshark* spočívá v převedení vstupního souboru do formátu PDML, který je možné zpracovat jako XML dokument, a není tedy zapotřebí implementovat disektory jednotlivých síťových protokolů. Při návrhu nástroje byl kladen největší důraz na jeho snadnou rozšiřitelnost o detekci nových síťových útoků, aniž by bylo nezbytné provádět výrazné změny v jeho implementaci. To je řešeno pomocí textových deklarativních zápisů síťových útoků, které je schopen nástroj interpretovat. Součástí práce je tedy i navržení obecného formátu pro deklarativní zápis síťových útoků, s jehož pomocí je možné popsat a následně detekovat specifikované síťové útoky.

Aby bylo možné deklarativní zápis síťových útoků navrhnout, bylo nejdříve zapotřebí identifikovat známé síťové útoky, které je možné detekovat ze zachycené síťové komunikace, nastudovat a popsat jejich principy a zranitelnosti, kterých tyto síťové útoky využívají. Na základě takto získaných informací pak navrhnout metody jejich detekce z paketů obsažených v zachycené síťové komunikaci. Pro tento účel bylo nezbytné všechny identifikované útoky realizovat v testovacím prostředí a zachytit síťovou komunikaci pomocí paketového analyzátoru v okamžiku jejich provádění. Zde bylo nutné řešit problém absence nástrojů pro generování některých typů útoků, který byl řešen použitím paketového manipulátoru Scapy, či získání síťové komunikace obsahující dané útoky z veřejných internetových databází. Následně proběhla hloubková analýza zachycené síťové komunikace převedené do formátu PDML, jejímž výsledkem byla identifikace metod pro detekci všech nastudovaných typů útoků na paketové úrovni.

Na základě takto získaných informací byl navrhnut obecný formát pro deklarativní zápis síťových útoků v serializačním formátu YAML, jehož hlavním účelem je poskytnutí základní rozšířitelnosti výsledné aplikace o nové síťové útoky. V něm jsou útoky rozděleny do tří typů, které se navzájem liší postupem aplikovaným při jejich detekci. Výsledkem tohoto kroku je tedy navržený formát pro deklarativní zápis síťových útoků, s jehož využitím byly vytvořeny konkrétní deklarativní zápisy všech identifikovaných útoků, jejichž interpretaci má vykonávat navrhovaný nástroj.

Dalším krokem byla samotná implementace nástroje, jejíž součástí bylo i navrhnutí několika optimalizací, které razantně snížily velikost PDML souborů, vytvořených konverzí vstupních souborů se zachycenou komunikací pomocí nástroje *tshark*. Těmito optimalizacemi byla výrazně redukována paměťová náročnost nástroje, díky tomu je nástroj schopen zpracovávat vstupní soubory o daleko větších velikostech, než byl schopen bez implementovaných optimalizací.

Implementované řešení bylo následně zapotřebí řádně otestovat a ověřit tak správný návrh formátu pro deklarativní zápis. První fáze testování byla zaměřena na funkční testování nástroje, které probíhalo na testovacích souborech obsahujících všechny identifikované síťové útoky. Tím bylo ověřeno, že nástroj je schopen korektně detekovat všechny typy útoků, pro které byl vytvořen jejich deklarativní zápis. V rámci funkčního testování bylo zapotřebí se zaměřit i na tzv. *false positives*, což jsou situace, kdy je za útok označen i běžný síťový provoz. Na základě těchto provedených testů pak docházelo k úpravám deklarativních zápisů i samotného formátu pro jejich zápis tak, aby došlo k maximální eliminaci falešných detekcí. V druhé fázi testování byla změřena výkonnost nástroje při zpracování vstupních souborů o různých velikostech. Na základě výkonnostních testů pak proběhlo vyhodnocení implementovaného nástroje.

Výsledkem diplomové práce je tedy program, který je schopen detekovat síťové útoky ze zachycené síťové komunikace pomocí deklarativních zápisů síťových útoků, jejichž návrh byl rovněž součástí této práce. Při potřebě rozšířit nástroj o detekci nového síťového útoku tedy postačuje vytvořit nový deklarativní zápis, který je nástroj schopen automatizovaně interpretovat. V současné verzi nástroje je vytvořeno celkem 23 různých deklarativních popisů.

Budoucí vývoj nástroje je možné věnovat rozšiřování prostředků, které navržený formát poskytuje pro deklarativní popis síťových útoků. To bude zapotřebí především v případech, kdy bude vyžadována podpora detekce nových síťových útoků, pro jejichž zápis jsou současné prostředky nedostatečné. Zároveň bude nezbytné reagovat na případné další falešné detekce, které nebyly odhaleny v provedeném testování, což povede k případné úpravě deklarativních zápisů síťových útoků, či přímo úpravě formátu pro jejich popis.

Literatura

[1] ARKKO, J.; KEMF, J.; ZILL, B.; aj.: SEcure Neighbor Discovery (SEND). RFC 3971, RFC Editor, Březen 2005.
URL https://tools.ietf.org/html/rfc3971

[2] BOUŠKA, P.: VLAN - Virtual Local Area Network. [Online]. 2007-06-02 [cit. 2017-12-28].
URL https://www.samuraj-cz.com/clanek/vlan-virtual-local-area-network/

[3] BRADEN, R.: Requirements for Internet Hosts - Communication Layers. STD 3, RFC Editor, Říjen 1989.
URL http://www.rfc-editor.org/rfc/rfc1122.txt

[4] CANAVAN, J. E.: *The Fundamentals of Network Security*. Artech House, 2001, ISBN 1580531768, s. 32 – 33.

[5] CARPENTER, B.; JIANG, S.: Significance of IPv6 Interface Identifiers. RFC 7136, RFC Editor, Únor 2014.
URL https://tools.ietf.org/html/rfc7136

[6] DEERING, S.; FENNER, W.; HABERMAN, B.: Multicast Listener Discovery (MLD) for IPv6. RFC 2710, RFC Editor, October 1999.
URL https://www.ietf.org/rfc/rfc2710.txt

[7] DROMS, R.: Dynamic Host Configuration Protocol. RFC 2131, RFC Editor, Březen 1997.
URL http://www.rfc-editor.org/rfc/rfc2131.txt

[8] ERICKSON, J.: *Hacking: The Art of Exploitation, 2nd Edition*. No Starch Press, 2008, ISBN 1593271441, s. 239–241.

[9] GONT, F.; CHOWN, T.: Network Reconnaissance in IPv6 Networks. RFC 7707, RFC Editor, Březen 2016.
URL https://tools.ietf.org/html/rfc7707

[10] GRAVES, K.: *CEH Certified Ethical Hacker Study Guide*. Sybex, 2010, ISBN 0470525207, str. 180.

[11] GREGG, M.; WATKINS, S.; MAYS, G.; aj.: *Hack the Stack: Using Snort and Ethereal to Master The 8 Layers of An Insecure Network*. Syngress, 2006, ISBN 1597491098, s. 122–123.

[12] HOGG, S.: Why You Must Use ICMPv6 Router Advertisements (RAs). [Online]. 2014-06-16 [cit. 2017-12-24].
URL https://community.infoblox.com/t5/IPv6-CoE-Blog/Why-You-Must-Use-ICMPv6-Router-Advertisements-RAs/ba-p/3416

[13] KAPIČÁK, D.: *Detekce skenování portů na vysokorychlostních sítích*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, květen 2014.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=16095&file=t

[14] KSHIRSAGAR, D.; RATHOD, A.; WATHORE, S.: Performance analysis of DoS LAND attack detection. *Perspectives in Science*, ročník 8, č. Supplement C, 2016: s. 736–738, ISSN 2213-0209.

[15] LOMNICKÝ, M.: *Analýza a demonstrace vybraných L2 útoků*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, květen 2009.

[16] MARIN, G.: Network security basics. *IEEE Security & Privacy*, ročník 3, č. 6, Prosinec 2005: s. 68–72, ISSN 1540-7993.
URL http://ieeexplore.ieee.org/document/1556540/

[17] McCLURE, S.; SCAMBRAY, J.; KURTZ, G.: *Hacking Exposed: Network Security Secrets and Solutions*. McGraw-Hill Education, sedmé vydání, 2012, ISBN 0071780289, 768 s.

[18] MUDALIAR, K.: *Performance Evaluation of Defence Mechanisms against ICMPv6 Router Advertisement Flood Attacks*. Diplomová práce, Unitec Institute of Technology, květen 2015.
URL http://unitec.researchbank.ac.nz/bitstream/handle/10652/3090/Keysha%20Mudaliar_2015-05-07.pdf?sequence=1&isAllowed=y

[19] OREBAUGH, A.; PINKARD, B.: *Nmap in the Enterprise: Your Guide to Network Scanning*. Syngress, 2008, ISBN 1597492418, 264 s.

[20] PLAŠIL, M.: *Soubor laboratorních úloh k demonstraci počítačových útoků*. Diplomová práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, květen 2015.
URL https://dspace.vutbr.cz/xmlui/bitstream/handle/11012/39969/DP-Pla%c5%a1il.pdf?sequence=2&isAllowed=y

[21] PLUMMER, D. C.: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. STD 37, RFC Editor, Listopad 1982.
URL http://www.rfc-editor.org/rfc/rfc826.txt

[22] POSTEL, J.: Internet Control Message Protocol. STD 5, RFC Editor, Září 1981.
URL http://www.rfc-editor.org/rfc/rfc792.txt

[23] POSTEL, J.: Internet Protocol. STD 5, RFC Editor, Září 1981.
URL http://www.rfc-editor.org/rfc/rfc791.txt

[24] POSTEL, J.: Transmission Control Protocol. STD 7, RFC Editor, Zaří 1981.
URL https://www.rfc-editor.org/rfc/rfc793.txt

[25] REHMAN, S. U.; MANICKAM, S.: Denial of Service Attack in IPv6 Duplicate Address Detection Process. *International Journal of Advanced Computer Science and Applications*, ročník 7, č. 6, 2016: s. 232–238, ISSN 2156-5570.
URL https://thesai.org/Downloads/Volume7No6/Paper_30-Denial_of_Service_Attack_in_IPv6_Duplicate_Address_Detection_Process.pdf

[26] SPANGLER, R.: Packet sniffing on layer 2 switched local area networks. *Packetwatch Research*, 2003.
URL http://www.packetwatch.net/documents/papers/layer2sniffing.pdf

[27] TULLOCH, M.: *Microsoft Encyclopedia of Security*. Microsoft Press, 2003, ISBN 0735618771, 414 s.

[28] VESELÝ, J.: *IPv6 a bezpečnost*. Diplomová práce, Masarykova univerzita v Brně, Fakulta informatiky, květen 2014.
URL https://is.muni.cz/th/pfa62/Vesely_DP.pdf

[29] YOUNES, O.: Securing ARP and DHCP for mitigating link layer attacks. *Sādhanā*, ročník 42, č. 12, Prosinec 2017: s. 2041–2053, ISSN 0973-7677.

[30] ZHANG, C.; XIONG, J.; WU, Q.: An efficient CGA algorithm against DoS attack on duplicate address detection process. In *2016 IEEE Wireless Communications and Networking Conference*, Duben 2016, s. 1–6, doi:10.1109/WCNC.2016.7564716.

[31] ZUZČÁK, M.: Bezpečnosť na LAN pod lupou: DHCP spoofing. [Online]. 2011-08-11 [cit. 2017-12-21].
URL
https://secit.sk/sk/content/bezpecnost-na-lan-pod-lupou-dhcp-spoofing

[32] ZUZČÁK, M.: Bezpečnosť na LAN pod lupou: Úvod a útok ARP cache poisoning. [Online]. 2011-07-05 [cit. 2017-12-21].
URL https://secit.sk/sk/content/bezpecnost-na-lan-pod-lupou-uvod-utok-arp-cache-poisoning

[33] ZUZČÁK, M.: Útok Router Advertisement flood - ako možno šikanovať užívateľov na IPv6 LAN. [Online]. 2014-09-10 [cit. 2017-12-25].
URL https://secit.sk/content/utok-router-advertisement-flood-ako-mozno-sikanovat-uzivatelov-na-ipv6-lan

[34] ČERNEKOVÁ, A.: *Analýza a demonstrace vybraných síťových útoků pod OS Linux*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, květen 2012.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=10731&file=t

Přílohy

Seznam příloh

Příloha A

Obsah DVD

DVD přiložené k této diplomové práci obsahuje:

- Zdrojové kódy textu diplomové práce v LaTeXu (adresář *text_dp_zdrojove_soubory*)
- Text diplomové práce v PDF formátu (adresář *text_dp_pdf*)
- Zdrojové kódy implementovaného nástroje (adresář *nastroj_zdrojove_soubory*)
- Deklarativní popisy síťových útoků (adresář *deklarativni_popisy*)
- Soubor testovacích dat obsahující zachycené síťové útoky (adresář *testovaci_data*)
- Instalační skripty pro instalaci potřebných knihoven (adresář *instalacni_skripty*)
- Soubor *README.md* popisující použití nástroje a tvorbu deklarativních zápisů útoků

Příloha B

Deklarativní zápis útoku LAND

```yaml
name: LAND
scope: atomic # Útok typu atomic
properties: # Sekce pro filtrování paketů
  - field-name: tcp.flags
    value: '0x00000002' # SYN paket
    valid: true
detection-conditions: # Sekce pro specifikaci podmínek detekce
  type: and
  conditions:
    - condition-type: expression # Podmínka typu expression
      expression: SrcIp == DstIp
      variables: # Specifikace hodnot proměnných ve výrazu
        - name: SrcIp # První proměnná
          type: field-value
          field-name: ip.src
          value-type: specific
        - name: DstIp # Druhá proměnná
          type: field-value
          field-name: ip.dst
          value-type: specific
threshold-error: 1 # Práh pro detekci
```

Příloha C

Deklarativní zápis skenování portů metodou TCP SYN

```
name: 'Port Scanning: SYN/TCP'
scope: stream # Typ útoku
properties: # Filtrování TCP spojení
  - field-name: tcp # Vyfiltrovat TCP spojeni
    valid: true
packets-specification: # Sekce pro specifikaci paketů v toku
  follow: true
  specified-only: false
  specification:
    - count: 1 # První paket v toku
      packet-properties:
        - field-name: tcp.flags
          value: '0x00000002' # SYN paket
          valid: true
    - count: 1 # Druhý paket v toku
      packet-properties:
        - field-name: tcp.flags
          value: '0x00000014' #RST ACK paket - znaci uzavreny port
          valid: true
threshold-warning: 200 # Prahy pro detekci a varování
threshold-error: 800
```

Příloha D

Deklarativní zápis útoku DAD

```
name: DAD attack
scope: group # Typ útoku
properties:
  - field-name: icmpv6 # Filtrování zpráv protokolu ICMPv6
    valid: true
packets-specification:
  follow: true
  specified-only: false
  specification: # Sekce pro specifikaci paketů
    - count: 1 # První typ paketu
      packet-properties:
        - field-name: icmpv6.type
          value: '135' # Filtrování Neighbour solicitation
          valid: true
        - field-name: ipv6.src
          value: "::" # Filtrování podle zdrojové IPv6 adresy
          valid: true
    - min-count: 1 # Druhý typ paketu
      max-count: "*"
      packet-properties:
        - field-name: icmpv6.type
          value: '136' # Filtrovaní Neighbour advertisement
          valid: true
        - field-name: ipv6.dst
          value: ff02::1 # Filtrování cílové IPv6 adresy
          valid: true
group-by: # Specifikace polí pro seskupení paketů
  - field-name: icmpv6.nd.ns.target_address
  - field-name: icmpv6.nd.na.target_address
threshold-warning: 1 # Prahy pro varovné a chybové hlášení
threshold-error: 5
```